

# The Java API XML and DOM Sources

Marco Mendieta Parihuancollo

Versión 036MPACAGAm2 (2025-10-18)

Secuencia 1: Source code

root: C:\Program Files\Java\jdk1.8.0\_131\src.zip

| Section                      | Package (49)                       | Files (513) | Lines (92909) |
|------------------------------|------------------------------------|-------------|---------------|
| Xml                          | javax.xml                          | 1           | 392           |
| Xml Bind                     | javax.xml.bind                     | 32          | 9120          |
| Xml Bind Annotation          | javax.xml.bind.annotation          | 35          | 4597          |
| Xml Bind Annotation Adapters | javax.xml.bind.annotation.adapters | 6           | 631           |
| Xml Bind Attachment          | javax.xml.bind.attachment          | 2           | 339           |
| Xml Bind Helpers             | javax.xml.bind.helpers             | 9           | 1863          |
| Xml Bind Util                | javax.xml.bind.util                | 4           | 618           |
| Xml Crypto                   | javax.xml.crypto                   | 14          | 1527          |
| Xml Crypto Dom               | javax.xml.crypto.dom               | 3           | 383           |
| Xml Crypto Dsig              | javax.xml.crypto.dsig              | 17          | 3005          |
| Xml Crypto Dsig Dom          | javax.xml.crypto.dsig.dom          | 2           | 344           |
| Xml Crypto Dsig Keyinfo      | javax.xml.crypto.dsig.keyinfo      | 8           | 1286          |
| Xml Crypto Dsig Spec         | javax.xml.crypto.dsig.spec         | 10          | 884           |
| Xml Datatype                 | javax.xml.datatype                 | 7           | 3894          |
| Xml Namespace                | javax.xml.namespace                | 2           | 804           |
| Xml Parsers                  | javax.xml.parsers                  | 8           | 2570          |
| Xml Soap                     | javax.xml.soap                     | 28          | 5688          |
| Xml Stream                   | javax.xml.stream                   | 17          | 4097          |
| Xml Stream Events            | javax.xml.stream.events            | 14          | 1045          |
| Xml Stream Util              | javax.xml.stream.util              | 4           | 561           |
| Xml Transform                | javax.xml.transform                | 14          | 2501          |
| Xml Transform Dom            | javax.xml.transform.dom            | 3           | 548           |
| Xml Transform Sax            | javax.xml.transform.sax            | 5           | 647           |
| Xml Transform Stax           | javax.xml.transform.stax           | 2           | 419           |
| Xml Transform Stream         | javax.xml.transform.stream         | 2           | 487           |
| Xml Validation               | javax.xml.validation               | 9           | 2869          |
| Xml Ws                       | javax.xml.ws                       | 31          | 3901          |
| Xml Ws Handler               | javax.xml.ws.handler               | 6           | 491           |
| Xml Ws Handler Soap          | javax.xml.ws.handler.soap          | 2           | 148           |
| Xml Ws Http                  | javax.xml.ws.http                  | 2           | 98            |
| Xml Ws Soap                  | javax.xml.ws.soap                  | 6           | 794           |
| Xml Ws Spi                   | javax.xml.ws.spi                   | 5           | 1536          |
| Xml Ws Spi Http              | javax.xml.ws.spi.http              | 4           | 576           |
| Xml Ws Wsaddressing          | javax.xml.ws.wsaddressing          | 3           | 548           |
| Xml Xpath                    | javax.xml.xpath                    | 13          | 2172          |
| Dom                          | org.w3c.dom                        | 28          | 5593          |

|                 |                         |    |      |
|-----------------|-------------------------|----|------|
| Dom Bootstrap   | org.w3c.dom.bootstrap   | 1  | 429  |
| Dom Css         | org.w3c.dom.css         | 22 | 3821 |
| Dom Events      | org.w3c.dom.events      | 8  | 930  |
| Dom Html        | org.w3c.dom.html        | 56 | 5174 |
| Dom Ls          | org.w3c.dom.ls          | 11 | 2113 |
| Dom Ranges      | org.w3c.dom.ranges      | 3  | 575  |
| Dom Stylesheets | org.w3c.dom.stylesheets | 5  | 440  |
| Dom Traversal   | org.w3c.dom.traversal   | 4  | 641  |
| Dom Views       | org.w3c.dom.views       | 2  | 115  |
| Dom Xpath       | org.w3c.dom.xpath       | 6  | 724  |
| Sax             | org.xml.sax             | 17 | 3966 |
| Sax Ext         | org.xml.sax.ext         | 8  | 1493 |
| Sax Helpers     | org.xml.sax.helpers     | 12 | 5512 |

---

Líneas de código: 92909 (1786 páginas)

## Contents

|          |                                           |           |
|----------|-------------------------------------------|-----------|
| <b>1</b> | <b>Xml</b>                                | <b>2</b>  |
| 1.1      | XMLConstants.java . . . . .               | 2         |
| <b>2</b> | <b>Xml Bind</b>                           | <b>10</b> |
| 2.1      | Binder.java . . . . .                     | 10        |
| 2.2      | ContextFinder.java . . . . .              | 18        |
| 2.3      | DataBindingException.java . . . . .       | 30        |
| 2.4      | DatatypeConverter.java . . . . .          | 31        |
| 2.5      | DatatypeConverterImpl.java . . . . .      | 45        |
| 2.6      | DatatypeConverterInterface.java . . . . . | 65        |
| 2.7      | Element.java . . . . .                    | 75        |
| 2.8      | GetPropertyAction.java . . . . .          | 76        |
| 2.9      | JAXB.java . . . . .                       | 77        |
| 2.10     | JAXBContext.java . . . . .                | 89        |
| 2.11     | JAXBElement.java . . . . .                | 104       |
| 2.12     | JAXBException.java . . . . .              | 108       |
| 2.13     | JAXBIntrospector.java . . . . .           | 112       |
| 2.14     | JAXBPermission.java . . . . .             | 114       |
| 2.15     | MarshalException.java . . . . .           | 115       |
| 2.16     | Marshaller.java . . . . .                 | 117       |
| 2.17     | Messages.java . . . . .                   | 133       |
| 2.18     | NotIdentifiableEvent.java . . . . .       | 135       |
| 2.19     | ParseConversionEvent.java . . . . .       | 136       |
| 2.20     | PrintConversionEvent.java . . . . .       | 137       |
| 2.21     | PropertyException.java . . . . .          | 138       |
| 2.22     | SchemaOutputResolver.java . . . . .       | 140       |
| 2.23     | TypeConstraintException.java . . . . .    | 142       |
| 2.24     | UnmarshalException.java . . . . .         | 145       |
| 2.25     | Unmarshaller.java . . . . .               | 147       |
| 2.26     | UnmarshallerHandler.java . . . . .        | 169       |
| 2.27     | ValidationEvent.java . . . . .            | 171       |
| 2.28     | ValidationEventHandler.java . . . . .     | 173       |
| 2.29     | ValidationEventLocator.java . . . . .     | 175       |
| 2.30     | ValidationException.java . . . . .        | 176       |

|          |                                        |            |
|----------|----------------------------------------|------------|
| 2.31     | Validator.java . . . . .               | 178        |
| 2.32     | WhiteSpaceProcessor.java . . . . .     | 184        |
| <b>3</b> | <b>Xml Bind Annotation</b>             | <b>188</b> |
| 3.1      | DomHandler.java . . . . .              | 188        |
| 3.2      | W3CDomHandler.java . . . . .           | 190        |
| 3.3      | XmlAccessOrder.java . . . . .          | 192        |
| 3.4      | XmlAccessType.java . . . . .           | 193        |
| 3.5      | XmlAccessorOrder.java . . . . .        | 194        |
| 3.6      | XmlAccessorType.java . . . . .         | 196        |
| 3.7      | XmlAnyAttribute.java . . . . .         | 198        |
| 3.8      | XmlAnyElement.java . . . . .           | 199        |
| 3.9      | XmlAttachmentRef.java . . . . .        | 205        |
| 3.10     | XmlAttribute.java . . . . .            | 207        |
| 3.11     | XmlElement.java . . . . .              | 209        |
| 3.12     | XmlElementDecl.java . . . . .          | 214        |
| 3.13     | XmlElementRef.java . . . . .           | 218        |
| 3.14     | XmlElementRefs.java . . . . .          | 223        |
| 3.15     | XmlElementWrapper.java . . . . .       | 224        |
| 3.16     | XmlElements.java . . . . .             | 227        |
| 3.17     | XmlEnum.java . . . . .                 | 231        |
| 3.18     | XmlEnumValue.java . . . . .            | 232        |
| 3.19     | XmlID.java . . . . .                   | 235        |
| 3.20     | XmlIDREF.java . . . . .                | 236        |
| 3.21     | XmlInlineBinaryData.java . . . . .     | 241        |
| 3.22     | XmlList.java . . . . .                 | 242        |
| 3.23     | XmlMimeType.java . . . . .             | 245        |
| 3.24     | XmlMixed.java . . . . .                | 246        |
| 3.25     | XmlNs.java . . . . .                   | 248        |
| 3.26     | XmlNsForm.java . . . . .               | 250        |
| 3.27     | XmlRegistry.java . . . . .             | 251        |
| 3.28     | XmlRootElement.java . . . . .          | 252        |
| 3.29     | XmlSchema.java . . . . .               | 255        |
| 3.30     | XmlSchemaType.java . . . . .           | 259        |
| 3.31     | XmlSchemaTypes.java . . . . .          | 261        |
| 3.32     | XmlSeeAlso.java . . . . .              | 263        |
| 3.33     | XmlTransient.java . . . . .            | 264        |
| 3.34     | XmlType.java . . . . .                 | 266        |
| 3.35     | XmlValue.java . . . . .                | 275        |
| <b>4</b> | <b>Xml Bind Annotation Adapters</b>    | <b>277</b> |
| 4.1      | CollapsedStringAdapter.java . . . . .  | 277        |
| 4.2      | HexBinaryAdapter.java . . . . .        | 280        |
| 4.3      | NormalizedStringAdapter.java . . . . . | 281        |
| 4.4      | XmlAdapter.java . . . . .              | 283        |
| 4.5      | XmlJavaTypeAdapter.java . . . . .      | 286        |
| 4.6      | XmlJavaTypeAdapters.java . . . . .     | 289        |
| <b>5</b> | <b>Xml Bind Attachment</b>             | <b>290</b> |
| 5.1      | AttachmentMarshaller.java . . . . .    | 290        |
| 5.2      | AttachmentUnmarshaller.java . . . . .  | 294        |

|           |                                    |            |
|-----------|------------------------------------|------------|
| <b>6</b>  | <b>Xml Bind Helpers</b>            | <b>297</b> |
| 6.1       | AbstractMarshallerImpl.java        | 297        |
| 6.2       | AbstractUnmarshallerImpl.java      | 306        |
| 6.3       | DefaultValidationEventHandler.java | 315        |
| 6.4       | Messages.java                      | 317        |
| 6.5       | NotIdentifiableEventImpl.java      | 319        |
| 6.6       | ParseConversionEventImpl.java      | 321        |
| 6.7       | PrintConversionEventImpl.java      | 323        |
| 6.8       | ValidationEventImpl.java           | 324        |
| 6.9       | ValidationEventLocatorImpl.java    | 328        |
| <b>7</b>  | <b>Xml Bind Util</b>               | <b>333</b> |
| 7.1       | JAXBResult.java                    | 333        |
| 7.2       | JAXBSource.java                    | 336        |
| 7.3       | Messages.java                      | 341        |
| 7.4       | ValidationEventCollector.java      | 343        |
| <b>8</b>  | <b>Xml Crypto</b>                  | <b>345</b> |
| 8.1       | AlgorithmMethod.java               | 345        |
| 8.2       | Data.java                          | 346        |
| 8.3       | KeySelector.java                   | 347        |
| 8.4       | KeySelectorException.java          | 350        |
| 8.5       | KeySelectorResult.java             | 353        |
| 8.6       | MarshalException.java              | 354        |
| 8.7       | NoSuchMechanismException.java      | 357        |
| 8.8       | NodeSetData.java                   | 360        |
| 8.9       | OctetStreamData.java               | 361        |
| 8.10      | URIDereferencer.java               | 363        |
| 8.11      | URIReference.java                  | 365        |
| 8.12      | URIReferenceException.java         | 366        |
| 8.13      | XMLCryptoContext.java              | 369        |
| 8.14      | XMLStructure.java                  | 374        |
| <b>9</b>  | <b>Xml Crypto Dom</b>              | <b>375</b> |
| 9.1       | DOMCryptoContext.java              | 375        |
| 9.2       | DOMStructure.java                  | 379        |
| 9.3       | DOMURIReference.java               | 381        |
| <b>10</b> | <b>Xml Crypto Dsig</b>             | <b>382</b> |
| 10.1      | CanonicalizationMethod.java        | 382        |
| 10.2      | DigestMethod.java                  | 384        |
| 10.3      | Manifest.java                      | 386        |
| 10.4      | Reference.java                     | 388        |
| 10.5      | SignatureMethod.java               | 391        |
| 10.6      | SignatureProperties.java           | 393        |
| 10.7      | SignatureProperty.java             | 395        |
| 10.8      | SignedInfo.java                    | 397        |
| 10.9      | Transform.java                     | 399        |
| 10.10     | TransformException.java            | 402        |
| 10.11     | TransformService.java              | 405        |
| 10.12     | XMLObject.java                     | 412        |
| 10.13     | XMLSignContext.java                | 415        |
| 10.14     | XMLSignature.java                  | 416        |



|                                                    |            |
|----------------------------------------------------|------------|
| 10.15 XMLSignatureException.java . . . . .         | 421        |
| 10.16 XMLSignatureFactory.java . . . . .           | 424        |
| 10.17 XMLValidateContext.java . . . . .            | 439        |
| <b>11 Xml Crypto Dsig Dom</b>                      | <b>440</b> |
| 11.1 DOMSignContext.java . . . . .                 | 440        |
| 11.2 DOMValidateContext.java . . . . .             | 444        |
| <b>12 Xml Crypto Dsig Keyinfo</b>                  | <b>447</b> |
| 12.1 KeyInfo.java . . . . .                        | 447        |
| 12.2 KeyInfoFactory.java . . . . .                 | 449        |
| 12.3 KeyName.java . . . . .                        | 460        |
| 12.4 KeyValue.java . . . . .                       | 461        |
| 12.5 PGPDData.java . . . . .                       | 464        |
| 12.6 RetrievalMethod.java . . . . .                | 466        |
| 12.7 X509Data.java . . . . .                       | 468        |
| 12.8 X509IssuerSerial.java . . . . .               | 470        |
| <b>13 Xml Crypto Dsig Spec</b>                     | <b>472</b> |
| 13.1 C14NMethodParameterSpec.java . . . . .        | 472        |
| 13.2 DigestMethodParameterSpec.java . . . . .      | 473        |
| 13.3 ExcC14NParameterSpec.java . . . . .           | 474        |
| 13.4 HMACParameterSpec.java . . . . .              | 476        |
| 13.5 SignatureMethodParameterSpec.java . . . . .   | 478        |
| 13.6 TransformParameterSpec.java . . . . .         | 479        |
| 13.7 XPathFilter2ParameterSpec.java . . . . .      | 479        |
| 13.8 XPathFilterParameterSpec.java . . . . .       | 481        |
| 13.9 XPathType.java . . . . .                      | 484        |
| 13.10 XSLTTransformParameterSpec.java . . . . .    | 488        |
| <b>14 Xml Datatype</b>                             | <b>490</b> |
| 14.1 DatatypeConfigurationException.java . . . . . | 490        |
| 14.2 DatatypeConstants.java . . . . .              | 491        |
| 14.3 DatatypeFactory.java . . . . .                | 497        |
| 14.4 Duration.java . . . . .                       | 519        |
| 14.5 FactoryFinder.java . . . . .                  | 538        |
| 14.6 SecuritySupport.java . . . . .                | 544        |
| 14.7 XMLGregorianCalendar.java . . . . .           | 546        |
| <b>15 Xml Namespace</b>                            | <b>567</b> |
| 15.1 NamespaceContext.java . . . . .               | 567        |
| 15.2 QName.java . . . . .                          | 573        |
| <b>16 Xml Parsers</b>                              | <b>582</b> |
| 16.1 DocumentBuilder.java . . . . .                | 582        |
| 16.2 DocumentBuilderFactory.java . . . . .         | 589        |
| 16.3 FactoryConfigurationError.java . . . . .      | 601        |
| 16.4 FactoryFinder.java . . . . .                  | 604        |
| 16.5 ParserConfigurationException.java . . . . .   | 610        |
| 16.6 SAXParser.java . . . . .                      | 611        |
| 16.7 SAXParserFactory.java . . . . .               | 621        |
| 16.8 SecuritySupport.java . . . . .                | 630        |

|                                               |            |
|-----------------------------------------------|------------|
| <b>17 Xml Soap</b>                            | <b>632</b> |
| 17.1 AttachmentPart.java . . . . .            | 632        |
| 17.2 Detail.java . . . . .                    | 642        |
| 17.3 DetailEntry.java . . . . .               | 644        |
| 17.4 FactoryFinder.java . . . . .             | 645        |
| 17.5 MessageFactory.java . . . . .            | 650        |
| 17.6 MimeHeader.java . . . . .                | 654        |
| 17.7 MimeHeaders.java . . . . .               | 655        |
| 17.8 Name.java . . . . .                      | 660        |
| 17.9 Node.java . . . . .                      | 662        |
| 17.10 SAAJMetaFactory.java . . . . .          | 664        |
| 17.11 SAAJResult.java . . . . .               | 667        |
| 17.12 SOAPBody.java . . . . .                 | 669        |
| 17.13 SOAPBodyElement.java . . . . .          | 675        |
| 17.14 SOAPConnection.java . . . . .           | 676        |
| 17.15 SOAPConnectionFactory.java . . . . .    | 678        |
| 17.16 SOAPConstants.java . . . . .            | 680        |
| 17.17 SOAPElement.java . . . . .              | 683        |
| 17.18 SOAPElementFactory.java . . . . .       | 693        |
| 17.19 SOAPEnvelope.java . . . . .             | 696        |
| 17.20 SOAPException.java . . . . .            | 700        |
| 17.21 SOAPFactory.java . . . . .              | 703        |
| 17.22 SOAPFault.java . . . . .                | 709        |
| 17.23 SOAPFaultElement.java . . . . .         | 719        |
| 17.24 SOAPHeader.java . . . . .               | 719        |
| 17.25 SOAPHeaderElement.java . . . . .        | 724        |
| 17.26 SOAPMessage.java . . . . .              | 728        |
| 17.27 SOAPPart.java . . . . .                 | 736        |
| 17.28 Text.java . . . . .                     | 742        |
| <b>18 Xml Stream</b>                          | <b>742</b> |
| 18.1 EventFilter.java . . . . .               | 742        |
| 18.2 FactoryConfigurationError.java . . . . . | 744        |
| 18.3 FactoryFinder.java . . . . .             | 746        |
| 18.4 Location.java . . . . .                  | 753        |
| 18.5 SecuritySupport.java . . . . .           | 755        |
| 18.6 StreamFilter.java . . . . .              | 757        |
| 18.7 XMLEventFactory.java . . . . .           | 758        |
| 18.8 XMLEventReader.java . . . . .            | 767        |
| 18.9 XMLEventWriter.java . . . . .            | 769        |
| 18.10 XMLInputFactory.java . . . . .          | 774        |
| 18.11 XMLOutputFactory.java . . . . .         | 784        |
| 18.12 XMLReporter.java . . . . .              | 791        |
| 18.13 XMLResolver.java . . . . .              | 792        |
| 18.14 XMLStreamConstants.java . . . . .       | 793        |
| 18.15 XMLStreamException.java . . . . .       | 796        |
| 18.16 XMLStreamReader.java . . . . .          | 798        |
| 18.17 XMLStreamWriter.java . . . . .          | 812        |

|                                                           |            |
|-----------------------------------------------------------|------------|
| <b>19 Xml Stream Events</b>                               | <b>822</b> |
| 19.1 Attribute.java . . . . .                             | 822        |
| 19.2 Characters.java . . . . .                            | 823        |
| 19.3 Comment.java . . . . .                               | 825        |
| 19.4 DTD.java . . . . .                                   | 826        |
| 19.5 EndDocument.java . . . . .                           | 827        |
| 19.6 EndElement.java . . . . .                            | 828        |
| 19.7 EntityDeclaration.java . . . . .                     | 829        |
| 19.8 EntityReference.java . . . . .                       | 831        |
| 19.9 Namespace.java . . . . .                             | 832        |
| 19.10 NotationDeclaration.java . . . . .                  | 833        |
| 19.11 ProcessingInstruction.java . . . . .                | 834        |
| 19.12 StartDocument.java . . . . .                        | 835        |
| 19.13 StartElement.java . . . . .                         | 837        |
| 19.14 XMLEvent.java . . . . .                             | 839        |
| <b>20 Xml Stream Util</b>                                 | <b>843</b> |
| 20.1 EventReaderDelegate.java . . . . .                   | 843        |
| 20.2 StreamReaderDelegate.java . . . . .                  | 845        |
| 20.3 XMLEventAllocator.java . . . . .                     | 851        |
| 20.4 XMLEventConsumer.java . . . . .                      | 852        |
| <b>21 Xml Transform</b>                                   | <b>854</b> |
| 21.1 ErrorListener.java . . . . .                         | 854        |
| 21.2 FactoryFinder.java . . . . .                         | 856        |
| 21.3 OutputKeys.java . . . . .                            | 863        |
| 21.4 Result.java . . . . .                                | 866        |
| 21.5 SecuritySupport.java . . . . .                       | 868        |
| 21.6 Source.java . . . . .                                | 870        |
| 21.7 SourceLocator.java . . . . .                         | 871        |
| 21.8 Templates.java . . . . .                             | 873        |
| 21.9 Transformer.java . . . . .                           | 875        |
| 21.10 TransformerConfigurationException.java . . . . .    | 881        |
| 21.11 TransformerException.java . . . . .                 | 883        |
| 21.12 TransformerFactory.java . . . . .                   | 890        |
| 21.13 TransformerFactoryConfigurationError.java . . . . . | 899        |
| 21.14 URIResolver.java . . . . .                          | 901        |
| <b>22 Xml Transform Dom</b>                               | <b>902</b> |
| 22.1 DOMLocator.java . . . . .                            | 902        |
| 22.2 DOMResult.java . . . . .                             | 903        |
| 22.3 DOMSource.java . . . . .                             | 911        |
| <b>23 Xml Transform Sax</b>                               | <b>913</b> |
| 23.1 SAXResult.java . . . . .                             | 913        |
| 23.2 SAXSource.java . . . . .                             | 916        |
| 23.3 SAXTransformerFactory.java . . . . .                 | 920        |
| 23.4 TemplatesHandler.java . . . . .                      | 923        |
| 23.5 TransformerHandler.java . . . . .                    | 924        |
| <b>24 Xml Transform Stax</b>                              | <b>926</b> |
| 24.1 StAXResult.java . . . . .                            | 926        |
| 24.2 StAXSource.java . . . . .                            | 929        |

|                                                     |             |
|-----------------------------------------------------|-------------|
| <b>25 Xml Transform Stream</b>                      | <b>934</b>  |
| 25.1 StreamResult.java . . . . .                    | 934         |
| 25.2 StreamSource.java . . . . .                    | 938         |
| <b>26 Xml Validation</b>                            | <b>943</b>  |
| 26.1 Schema.java . . . . .                          | 943         |
| 26.2 SchemaFactory.java . . . . .                   | 945         |
| 26.3 SchemaFactoryConfigurationError.java . . . . . | 961         |
| 26.4 SchemaFactoryFinder.java . . . . .             | 963         |
| 26.5 SchemaFactoryLoader.java . . . . .             | 971         |
| 26.6 SecuritySupport.java . . . . .                 | 973         |
| 26.7 TypeInfoProvider.java . . . . .                | 976         |
| 26.8 Validator.java . . . . .                       | 980         |
| 26.9 ValidatorHandler.java . . . . .                | 990         |
| <b>27 Xml Ws</b>                                    | <b>999</b>  |
| 27.1 Action.java . . . . .                          | 999         |
| 27.2 AsyncHandler.java . . . . .                    | 1002        |
| 27.3 Binding.java . . . . .                         | 1003        |
| 27.4 BindingProvider.java . . . . .                 | 1004        |
| 27.5 BindingType.java . . . . .                     | 1008        |
| 27.6 Dispatch.java . . . . .                        | 1009        |
| 27.7 Endpoint.java . . . . .                        | 1011        |
| 27.8 EndpointContext.java . . . . .                 | 1021        |
| 27.9 EndpointReference.java . . . . .               | 1022        |
| 27.10 FaultAction.java . . . . .                    | 1026        |
| 27.11 Holder.java . . . . .                         | 1029        |
| 27.12 LogicalMessage.java . . . . .                 | 1030        |
| 27.13 ProtocolException.java . . . . .              | 1032        |
| 27.14 Provider.java . . . . .                       | 1034        |
| 27.15 RequestWrapper.java . . . . .                 | 1035        |
| 27.16 RespectBinding.java . . . . .                 | 1036        |
| 27.17 RespectBindingFeature.java . . . . .          | 1038        |
| 27.18 Response.java . . . . .                       | 1040        |
| 27.19 ResponseWrapper.java . . . . .                | 1041        |
| 27.20 Service.java . . . . .                        | 1043        |
| 27.21 ServiceMode.java . . . . .                    | 1057        |
| 27.22 WebEndpoint.java . . . . .                    | 1058        |
| 27.23 WebFault.java . . . . .                       | 1059        |
| 27.24 WebServiceClient.java . . . . .               | 1061        |
| 27.25 WebServiceContext.java . . . . .              | 1062        |
| 27.26 WebServiceException.java . . . . .            | 1065        |
| 27.27 WebServiceFeature.java . . . . .              | 1066        |
| 27.28 WebServicePermission.java . . . . .           | 1068        |
| 27.29 WebServiceProvider.java . . . . .             | 1070        |
| 27.30 WebServiceRef.java . . . . .                  | 1071        |
| 27.31 WebServiceRefs.java . . . . .                 | 1074        |
| <b>28 Xml Ws Handler</b>                            | <b>1075</b> |
| 28.1 Handler.java . . . . .                         | 1075        |
| 28.2 HandlerResolver.java . . . . .                 | 1077        |
| 28.3 LogicalHandler.java . . . . .                  | 1078        |
| 28.4 LogicalMessageContext.java . . . . .           | 1079        |

|           |                                         |             |
|-----------|-----------------------------------------|-------------|
| 28.5      | MessageContext.java                     | 1080        |
| 28.6      | PortInfo.java                           | 1084        |
| <b>29</b> | <b>Xml Ws Handler Soap</b>              | <b>1085</b> |
| 29.1      | SOAPHandler.java                        | 1085        |
| 29.2      | SOAPMessageContext.java                 | 1086        |
| <b>30</b> | <b>Xml Ws Http</b>                      | <b>1088</b> |
| 30.1      | HTTPBinding.java                        | 1088        |
| 30.2      | HTTPException.java                      | 1089        |
| <b>31</b> | <b>Xml Ws Soap</b>                      | <b>1090</b> |
| 31.1      | Addressing.java                         | 1090        |
| 31.2      | AddressingFeature.java                  | 1092        |
| 31.3      | MTOM.java                               | 1098        |
| 31.4      | MTOMFeature.java                        | 1099        |
| 31.5      | SOAPBinding.java                        | 1102        |
| 31.6      | SOAPFaultException.java                 | 1104        |
| <b>32</b> | <b>Xml Ws Spi</b>                       | <b>1106</b> |
| 32.1      | FactoryFinder.java                      | 1106        |
| 32.2      | Invoker.java                            | 1110        |
| 32.3      | Provider.java                           | 1112        |
| 32.4      | ServiceDelegate.java                    | 1122        |
| 32.5      | WebServiceFeatureAnnotation.java        | 1134        |
| <b>33</b> | <b>Xml Ws Spi Http</b>                  | <b>1135</b> |
| 33.1      | HttpContext.java                        | 1135        |
| 33.2      | HttpExchange.java                       | 1137        |
| 33.3      | HttpHandler.java                        | 1144        |
| 33.4      | package-info.java                       | 1145        |
| <b>34</b> | <b>Xml Ws Wsaddressing</b>              | <b>1147</b> |
| 34.1      | W3CEndpointReference.java               | 1147        |
| 34.2      | W3CEndpointReferenceBuilder.java        | 1150        |
| 34.3      | package-info.java                       | 1157        |
| <b>35</b> | <b>Xml Xpath</b>                        | <b>1157</b> |
| 35.1      | SecuritySupport.java                    | 1157        |
| 35.2      | XPath.java                              | 1160        |
| 35.3      | XPathConstants.java                     | 1167        |
| 35.4      | XPathException.java                     | 1169        |
| 35.5      | XPathExpression.java                    | 1172        |
| 35.6      | XPathExpressionException.java           | 1176        |
| 35.7      | XPathFactory.java                       | 1178        |
| 35.8      | XPathFactoryConfigurationException.java | 1186        |
| 35.9      | XPathFactoryFinder.java                 | 1187        |
| 35.10     | XPathFunction.java                      | 1196        |
| 35.11     | XPathFunctionException.java             | 1197        |
| 35.12     | XPathFunctionResolver.java              | 1198        |
| 35.13     | XPathVariableResolver.java              | 1200        |

|                                     |             |
|-------------------------------------|-------------|
| <b>36 Dom</b>                       | <b>1201</b> |
| 36.1 Attr.java                      | 1201        |
| 36.2 CDATASection.java              | 1207        |
| 36.3 CharacterData.java             | 1209        |
| 36.4 Comment.java                   | 1212        |
| 36.5 DOMConfiguration.java          | 1213        |
| 36.6 DOMError.java                  | 1222        |
| 36.7 DOMErrorHandler.java           | 1225        |
| 36.8 DOMException.java              | 1226        |
| 36.9 DOMImplementation.java         | 1229        |
| 36.10 DOMImplementationList.java    | 1233        |
| 36.11 DOMImplementationSource.java  | 1234        |
| 36.12 DOMLocator.java               | 1236        |
| 36.13 DOMStringList.java            | 1238        |
| 36.14 Document.java                 | 1239        |
| 36.15 DocumentFragment.java         | 1256        |
| 36.16 DocumentType.java             | 1258        |
| 36.17 Element.java                  | 1260        |
| 36.18 Entity.java                   | 1269        |
| 36.19 EntityReference.java          | 1271        |
| 36.20 NameList.java                 | 1273        |
| 36.21 NamedNodeMap.java             | 1275        |
| 36.22 Node.java                     | 1279        |
| 36.23 NodeList.java                 | 1297        |
| 36.24 Notation.java                 | 1299        |
| 36.25 ProcessingInstruction.java    | 1300        |
| 36.26 Text.java                     | 1302        |
| 36.27 TypeInfo.java                 | 1305        |
| 36.28 UserDataHandler.java          | 1310        |
| <b>37 Dom Bootstrap</b>             | <b>1312</b> |
| 37.1 DOMImplementationRegistry.java | 1312        |
| <b>38 Dom Css</b>                   | <b>1320</b> |
| 38.1 CSS2Properties.java            | 1320        |
| 38.2 CSSCharsetRule.java            | 1354        |
| 38.3 CSSFontFaceRule.java           | 1356        |
| 38.4 CSSImportRule.java             | 1357        |
| 38.5 CSSMediaRule.java              | 1358        |
| 38.6 CSSPageRule.java               | 1361        |
| 38.7 CSSPrimitiveValue.java         | 1362        |
| 38.8 CSSRule.java                   | 1368        |
| 38.9 CSSRuleList.java               | 1371        |
| 38.10 CSSStyleDeclaration.java      | 1372        |
| 38.11 CSSStyleRule.java             | 1376        |
| 38.12 CSSStyleSheet.java            | 1377        |
| 38.13 CSSUnknownRule.java           | 1380        |
| 38.14 CSSValue.java                 | 1381        |
| 38.15 CSSValueList.java             | 1383        |
| 38.16 Counter.java                  | 1384        |
| 38.17 DOMImplementationCSS.java     | 1386        |
| 38.18 DocumentCSS.java              | 1387        |
| 38.19 ElementCSSInlineStyle.java    | 1389        |

|           |                            |             |
|-----------|----------------------------|-------------|
| 38.20     | RGBColor.java              | 1390        |
| 38.21     | Rect.java                  | 1391        |
| 38.22     | ViewCSS.java               | 1393        |
| <b>39</b> | <b>Dom Events</b>          | <b>1394</b> |
| 39.1      | DocumentEvent.java         | 1394        |
| 39.2      | Event.java                 | 1396        |
| 39.3      | EventException.java        | 1399        |
| 39.4      | EventListener.java         | 1401        |
| 39.5      | EventTarget.java           | 1402        |
| 39.6      | MouseEvent.java            | 1405        |
| 39.7      | MutationEvent.java         | 1408        |
| 39.8      | UIEvent.java               | 1411        |
| <b>40</b> | <b>Dom Html</b>            | <b>1413</b> |
| 40.1      | HTMLAnchorElement.java     | 1413        |
| 40.2      | HTMLAppletElement.java     | 1415        |
| 40.3      | HTMLAreaElement.java       | 1418        |
| 40.4      | HTMLBRElement.java         | 1420        |
| 40.5      | HTMLBaseElement.java       | 1421        |
| 40.6      | HTMLBaseFontElement.java   | 1422        |
| 40.7      | HTMLBodyElement.java       | 1424        |
| 40.8      | HTMLButtonElement.java     | 1426        |
| 40.9      | HTMLCollection.java        | 1428        |
| 40.10     | HTMLDListElement.java      | 1429        |
| 40.11     | HTMLDOMImplementation.java | 1431        |
| 40.12     | HTMLDirectoryElement.java  | 1432        |
| 40.13     | HTMLDivElement.java        | 1433        |
| 40.14     | HTMLDocument.java          | 1434        |
| 40.15     | HTMLElement.java           | 1438        |
| 40.16     | HTMLFieldSetElement.java   | 1440        |
| 40.17     | HTMLFontElement.java       | 1441        |
| 40.18     | HTMLFormElement.java       | 1442        |
| 40.19     | HTMLFrameElement.java      | 1444        |
| 40.20     | HTMLFrameSetElement.java   | 1447        |
| 40.21     | HTMLHRElement.java         | 1448        |
| 40.22     | HTMLHeadElement.java       | 1449        |
| 40.23     | HTMLHeadingElement.java    | 1451        |
| 40.24     | HTMLHtmlElement.java       | 1452        |
| 40.25     | HTMLIFrameElement.java     | 1453        |
| 40.26     | HTMLImageElement.java      | 1455        |
| 40.27     | HTMLInputElement.java      | 1458        |
| 40.28     | HTMLIsIndexElement.java    | 1462        |
| 40.29     | HTMLLIElement.java         | 1464        |
| 40.30     | HTMLLabelElement.java      | 1465        |
| 40.31     | HTMLLegendElement.java     | 1466        |
| 40.32     | HTMLLinkElement.java       | 1468        |
| 40.33     | HTMLMapElement.java        | 1470        |
| 40.34     | HTMLMenuElement.java       | 1471        |
| 40.35     | HTMLMetaElement.java       | 1473        |
| 40.36     | HTMLModElement.java        | 1474        |
| 40.37     | HTMLOLListElement.java     | 1475        |
| 40.38     | HTMLObjectElement.java     | 1477        |

|           |                              |             |
|-----------|------------------------------|-------------|
| 40.39     | HTMLOptGroupElement.java     | 1480        |
| 40.40     | HTMLOptionElement.java       | 1482        |
| 40.41     | HTMLParagraphElement.java    | 1484        |
| 40.42     | HTMLParamElement.java        | 1485        |
| 40.43     | HTMLPreElement.java          | 1487        |
| 40.44     | HTMLQuoteElement.java        | 1488        |
| 40.45     | HTMLScriptElement.java       | 1489        |
| 40.46     | HTMLSelectElement.java       | 1491        |
| 40.47     | HTMLStyleElement.java        | 1494        |
| 40.48     | HTMLTableCaptionElement.java | 1495        |
| 40.49     | HTMLTableCellElement.java    | 1496        |
| 40.50     | HTMLTableColElement.java     | 1499        |
| 40.51     | HTMLTableElement.java        | 1501        |
| 40.52     | HTMLTableRowElement.java     | 1505        |
| 40.53     | HTMLTableSectionElement.java | 1508        |
| 40.54     | HTMLTextAreaElement.java     | 1510        |
| 40.55     | HTMLTitleElement.java        | 1513        |
| 40.56     | HTMLULListElement.java       | 1514        |
| <b>41</b> | <b>Dom Ls</b>                | <b>1516</b> |
| 41.1      | DOMImplementationLS.java     | 1516        |
| 41.2      | LSEException.java            | 1519        |
| 41.3      | LSInput.java                 | 1520        |
| 41.4      | LSLoadEvent.java             | 1525        |
| 41.5      | LSOutput.java                | 1526        |
| 41.6      | LSParser.java                | 1529        |
| 41.7      | LSParserFilter.java          | 1539        |
| 41.8      | LSProgressEvent.java         | 1543        |
| 41.9      | LSResourceResolver.java      | 1545        |
| 41.10     | LSSerializer.java            | 1547        |
| 41.11     | LSSerializerFilter.java      | 1556        |
| <b>42</b> | <b>Dom Ranges</b>            | <b>1558</b> |
| 42.1      | DocumentRange.java           | 1558        |
| 42.2      | Range.java                   | 1560        |
| 42.3      | RangeException.java          | 1568        |
| <b>43</b> | <b>Dom Stylesheets</b>       | <b>1570</b> |
| 43.1      | DocumentStyle.java           | 1570        |
| 43.2      | LinkStyle.java               | 1571        |
| 43.3      | MediaList.java               | 1572        |
| 43.4      | StyleSheet.java              | 1574        |
| 43.5      | StyleSheetList.java          | 1577        |
| <b>44</b> | <b>Dom Traversal</b>         | <b>1578</b> |
| 44.1      | DocumentTraversal.java       | 1578        |
| 44.2      | NodeFilter.java              | 1581        |
| 44.3      | NodeIterator.java            | 1584        |
| 44.4      | TreeWalker.java              | 1587        |
| <b>45</b> | <b>Dom Views</b>             | <b>1591</b> |
| 45.1      | AbstractView.java            | 1591        |
| 45.2      | DocumentView.java            | 1592        |



|                                                |                 |
|------------------------------------------------|-----------------|
| <b>46 Dom Xpath</b>                            | <b>1593</b>     |
| 46.1 XPathEvaluator.java . . . . .             | 1593            |
| 46.2 XPathException.java . . . . .             | 1596            |
| 46.3 XPathExpression.java . . . . .            | 1598            |
| 46.4 XPathNSResolver.java . . . . .            | 1600            |
| 46.5 XPathNamespace.java . . . . .             | 1601            |
| 46.6 XPathResult.java . . . . .                | 1603            |
| <br><b>47 Sax</b>                              | <br><b>1608</b> |
| 47.1 AttributeList.java . . . . .              | 1608            |
| 47.2 Attributes.java . . . . .                 | 1612            |
| 47.3 ContentHandler.java . . . . .             | 1617            |
| 47.4 DTDHandler.java . . . . .                 | 1626            |
| 47.5 DocumentHandler.java . . . . .            | 1628            |
| 47.6 EntityResolver.java . . . . .             | 1633            |
| 47.7 ErrorHandler.java . . . . .               | 1636            |
| 47.8 HandlerBase.java . . . . .                | 1639            |
| 47.9 InputSource.java . . . . .                | 1647            |
| 47.10 Locator.java . . . . .                   | 1653            |
| 47.11 Parser.java . . . . .                    | 1657            |
| 47.12 SAXException.java . . . . .              | 1661            |
| 47.13 SAXNotRecognizedException.java . . . . . | 1665            |
| 47.14 SAXNotSupportedException.java . . . . .  | 1666            |
| 47.15 SAXParseException.java . . . . .         | 1668            |
| 47.16 XMLFilter.java . . . . .                 | 1674            |
| 47.17 XMLReader.java . . . . .                 | 1676            |
| <br><b>48 Sax Ext</b>                          | <br><b>1684</b> |
| 48.1 Attributes2.java . . . . .                | 1684            |
| 48.2 Attributes2Impl.java . . . . .            | 1687            |
| 48.3 DeclHandler.java . . . . .                | 1693            |
| 48.4 DefaultHandler2.java . . . . .            | 1696            |
| 48.5 EntityResolver2.java . . . . .            | 1699            |
| 48.6 LexicalHandler.java . . . . .             | 1704            |
| 48.7 Locator2.java . . . . .                   | 1708            |
| 48.8 Locator2Impl.java . . . . .               | 1710            |
| <br><b>49 Sax Helpers</b>                      | <br><b>1713</b> |
| 49.1 AttributeListImpl.java . . . . .          | 1713            |
| 49.2 AttributesImpl.java . . . . .             | 1719            |
| 49.3 DefaultHandler.java . . . . .             | 1731            |
| 49.4 LocatorImpl.java . . . . .                | 1741            |
| 49.5 NamespaceSupport.java . . . . .           | 1745            |
| 49.6 NewInstance.java . . . . .                | 1761            |
| 49.7 ParserAdapter.java . . . . .              | 1763            |
| 49.8 ParserFactory.java . . . . .              | 1783            |
| 49.9 SecuritySupport.java . . . . .            | 1786            |
| 49.10 XMLFilterImpl.java . . . . .             | 1788            |
| 49.11 XMLReaderAdapter.java . . . . .          | 1802            |
| 49.12 XMLReaderFactory.java . . . . .          | 1813            |

# 1 Xml (javax.xml package)

## 1.1 XMLConstants.java

Secuencia 2: javax.xml/XMLConstants.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml;
27
28 /**
29  * <p>Utility class to contain basic XML values as constants.</p>
30  *
31  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
32  * @version $Revision: 1.8 $, $Date: 2010/05/25 16:19:45 $
33  * @see <a href="http://www.w3.org/TR/xml11/">Extensible Markup Language (XML) 1.1</a>
34  * @see <a href="http://www.w3.org/TR/REC-xml">Extensible Markup Language (XML) 1.0 (Second Edition
35  *    )</a>
36  * @see <a href="http://www.w3.org/XML/xml-V10-2e-errata">XML 1.0 Second Edition Specification
37  *    Errata</a>
38  * @see <a href="http://www.w3.org/TR/xml-names11/">Namespaces in XML 1.1</a>
39  * @see <a href="http://www.w3.org/TR/REC-xml-names">Namespaces in XML</a>
40  * @see <a href="http://www.w3.org/XML/xml-names-19990114-errata">Namespaces in XML Errata</a>
41  * @see <a href="http://www.w3.org/TR/xmlschema-1/">XML Schema Part 1: Structures</a>
42  * @since 1.5
43  */
44
45 public final class XMLConstants {

```

```
46     * <p>Private constructor to prevent instantiation.</p>
47     */
48     private XMLConstants() {
49     }
50
51     /**
52     * <p>Namespace URI to use to represent that there is no Namespace.</p>
53     *
54     * <p>Defined by the Namespace specification to be "".</p>
55     *
56     * @see <a href="http://www.w3.org/TR/REC-xml-names/#defaulting">
57     * Namespaces in XML, 5.2 Namespace Defaulting</a>
58     */
59     public static final String NULL_NS_URI = "";
60
61     /**
62     * <p>Prefix to use to represent the default XML Namespace.</p>
63     *
64     * <p>Defined by the XML specification to be "".</p>
65     *
66     * @see <a
67     * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">
68     * Namespaces in XML, 3. Qualified Names</a>
69     */
70     public static final String DEFAULT_NS_PREFIX = "";
71
72     /**
73     * <p>The official XML Namespace name URI.</p>
74     *
75     * <p>Defined by the XML specification to be
76     * "{@code http://www.w3.org/XML/1998/namespace}".</p>
77     *
78     * @see <a
79     * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">
80     * Namespaces in XML, 3. Qualified Names</a>
81     */
82     public static final String XML_NS_URI =
83         "http://www.w3.org/XML/1998/namespace";
84
85     /**
86     * <p>The official XML Namespace prefix.</p>
87     *
88     * <p>Defined by the XML specification to be "{@code xml}".</p>
89     *
90     * @see <a
91     * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">
92     * Namespaces in XML, 3. Qualified Names</a>
93     */
94     public static final String XML_NS_PREFIX = "xml";
95
96     /**
97     * <p>The official XML attribute used for specifying XML Namespace
98     * declarations, {@link #XMLNS_ATTRIBUTE
```

```

99      * XMLConstants.XMLNS_ATTRIBUTE}, Namespace name URI.</p>
100      *
101      * <p>Defined by the XML specification to be
102      * "{@code http://www.w3.org/2000/xmlns/}"</p>
103      *
104      * @see <a
105      * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">
106      * Namespaces in XML, 3. Qualified Names</a>
107      * @see <a
108      * href="http://www.w3.org/XML/xml-names-19990114-errata">
109      * Namespaces in XML Errata</a>
110      */
111     public static final String XMLNS_ATTRIBUTE_NS_URI =
112         "http://www.w3.org/2000/xmlns/";
113
114     /**
115      * <p>The official XML attribute used for specifying XML Namespace
116      * declarations.</p>
117      *
118      * <p>It is <strong><em>NOT</em></strong> valid to use as a
119      * prefix. Defined by the XML specification to be
120      * "{@code xmlns}"</p>
121      *
122      * @see <a
123      * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">
124      * Namespaces in XML, 3. Qualified Names</a>
125      */
126     public static final String XMLNS_ATTRIBUTE = "xmlns";
127
128     /**
129      * <p>W3C XML Schema Namespace URI.</p>
130      *
131      * <p>Defined to be "{@code http://www.w3.org/2001/XMLSchema}"</p>
132      *
133      * @see <a href=
134      * "http://www.w3.org/TR/xmlschema-1/#Instance_Document_Constructions">
135      * XML Schema Part 1:
136      * Structures, 2.6 Schema-Related Markup in Documents Being Validated</a>
137      */
138     public static final String W3C_XML_SCHEMA_NS_URI =
139         "http://www.w3.org/2001/XMLSchema";
140
141     /**
142      * <p>W3C XML Schema Instance Namespace URI.</p>
143      *
144      * <p>Defined to be "{@code http://www.w3.org/2001/XMLSchema-instance}"</p>
145      *
146      * @see <a href=
147      * "http://www.w3.org/TR/xmlschema-1/#Instance_Document_Constructions">
148      * XML Schema Part 1:
149      * Structures, 2.6 Schema-Related Markup in Documents Being Validated</a>
150      */
151     public static final String W3C_XML_SCHEMA_INSTANCE_NS_URI =

```

```

152     "http://www.w3.org/2001/XMLSchema-instance";
153
154     /**
155      * <p>W3C XPath Datatype Namespace URI.</p>
156      *
157      * <p>Defined to be "{@code http://www.w3.org/2003/11/xpath-datatypes}".</p>
158      *
159      * @see <a href="http://www.w3.org/TR/xpath-datamodel">XQuery 1.0 and XPath 2.0 Data Model
160      *      </a>
161      */
162     public static final String W3C_XPATH_DATATYPE_NS_URI = "http://www.w3.org/2003/11/xpath-
163         datatypes";
164
165     /**
166      * <p>XML Document Type Declaration Namespace URI as an arbitrary value.</p>
167      *
168      * <p>Since not formally defined by any existing standard, arbitrarily define to be "{@code
169      *      http://www.w3.org/TR/REC-xml}".
170      */
171     public static final String XML_DTD_NS_URI = "http://www.w3.org/TR/REC-xml";
172
173     /**
174      * <p>RELAX NG Namespace URI.</p>
175      *
176      * <p>Defined to be "{@code http://relaxng.org/ns/structure/1.0}".</p>
177      *
178      * @see <a href="http://relaxng.org/spec-20011203.html">RELAX NG Specification</a>
179      */
180     public static final String RELAXNG_NS_URI = "http://relaxng.org/ns/structure/1.0";
181
182     /**
183      * <p>Feature for secure processing.</p>
184      *
185      * <ul>
186      * <li>
187      *      {@code true} instructs the implementation to process XML securely.
188      *      This may set limits on XML constructs to avoid conditions such as denial of service
189      *      attacks.
190      * </li>
191      * <li>
192      *      {@code false} instructs the implementation to process XML in accordance with the XML
193      *      specifications
194      *      ignoring security issues such as limits on XML constructs to avoid conditions such
195      *      as denial of service attacks.
196      * </li>
197      * </ul>
198      */
199     public static final String FEATURE_SECURE_PROCESSING = "http://javax.xml.XMLConstants/
200         feature/secure-processing";
201
202     /**
203      * <p>Property: accessExternalDTD</p>

```

```

198      *
199      * <p>
200      * Restrict access to external DTDs and external Entity References to the protocols
        specified.
201      * If access is denied due to the restriction of this property, a runtime exception that
202      * is specific to the context is thrown. In the case of {@link javax.xml.parsers.SAXParser}
203      * for example, {@link org.xml.sax.SAXException} is thrown.
204      * </p>
205      *
206      * <p>
207      * <b>Value: </b> a list of protocols separated by comma. A protocol is the scheme portion
        of a
208      * {@link java.net.URI}, or in the case of the JAR protocol, "jar" plus the scheme portion
209      * separated by colon.
210      * A scheme is defined as:
211      *
212      * <blockquote>
213      * scheme = alpha *( alpha | digit | "+" | "-" | "." )<br>
214      * where alpha = a-z and A-Z.<br><br>
215      *
216      * And the JAR protocol:<br>
217      *
218      * jar[:scheme]<br><br>
219      *
220      * Protocols including the keyword "jar" are case-insensitive. Any whitespaces as defined
        by
221      * {@link java.lang.Character#isSpaceChar } in the value will be ignored.
222      * Examples of protocols are file, http, jar:file.
223      *
224      * </blockquote>
225      *</p>
226      *
227      *<p>
228      * <b>Default value:</b> The default value is implementation specific and therefore not
        specified.
229      * The following options are provided for consideration:
230      * <blockquote>
231      * <UL>
232      *     <LI>an empty string to deny all access to external references;</LI>
233      *     <LI>a specific protocol, such as file, to give permission to only the protocol;</LI>
234      *     <LI>the keyword "all" to grant permission to all protocols.</LI>
235      * </UL><br>
236      *     When FEATURE_SECURE_PROCESSING is enabled, it is recommended that implementations
237      *     restrict external connections by default, though this may cause problems for
        applications
238      *     that process XML/XSD/XSL with external references.
239      * </blockquote>
240      * </p>
241      *
242      * <p>
243      * <b>Granting all access:</b> the keyword "all" grants permission to all protocols.
244      * </p>
245      * <p>

```

```

246      * <b>System Property:</b> The value of this property can be set or overridden by
247      * system property {@code javax.xml.accessExternalDTD}.
248      * </p>
249      *
250      * <p>
251      * <b>${JAVA_HOME}/lib/jaxp.properties:</b> This configuration file is in standard
252      * {@link java.util.Properties} format. If the file exists and the system property is
253      * specified,
254      * its value will be used to override the default of the property.
255      * </p>
256      *
257      * <p>
258      * </p>
259      * @since 1.7
260      */
261      public static final String ACCESS_EXTERNAL_DTD = "http://javax.xml.XMLConstants/property/
262          accessExternalDTD";
263
264      /**
265      * <p>Property: accessExternalSchema</p>
266      *
267      * <p>
268      * Restrict access to the protocols specified for external reference set by the
269      * schemaLocation attribute, Import and Include element. If access is denied
270      * due to the restriction of this property, a runtime exception that is specific
271      * to the context is thrown. In the case of {@link javax.xml.validation.SchemaFactory}
272      * for example, org.xml.sax.SAXException is thrown.
273      * </p>
274      * <p>
275      * <b>Value:</b> a list of protocols separated by comma. A protocol is the scheme portion
276      * of a
277      * {@link java.net.URI}, or in the case of the JAR protocol, "jar" plus the scheme portion
278      * separated by colon.
279      * A scheme is defined as:
280      *
281      * <blockquote>
282      * scheme = alpha *( alpha | digit | "+" | "-" | "." )<br>
283      * where alpha = a-z and A-Z.<br><br>
284      *
285      * And the JAR protocol:<br>
286      *
287      * jar[:scheme]<br><br>
288      * Protocols including the keyword "jar" are case-insensitive. Any whitespaces as defined
289      * by
290      * {@link java.lang.Character#isSpaceChar } in the value will be ignored.
291      * Examples of protocols are file, http, jar:file.
292      *
293      * </blockquote>
294      * </p>

```

```

295      * <b>Default value:</b> The default value is implementation specific and therefore not
      * specified.
296      * The following options are provided for consideration:
297      * <blockquote>
298      * <UL>
299      *     <LI>an empty string to deny all access to external references;</LI>
300      *     <LI>a specific protocol, such as file, to give permission to only the protocol;</LI>
301      *     <LI>the keyword "all" to grant permission to all protocols.</LI>
302      * </UL><br>
303      *     When FEATURE_SECURE_PROCESSING is enabled, it is recommended that implementations
304      *     restrict external connections by default, though this may cause problems for
      *     applications
305      *     that process XML/XSD/XSL with external references.
306      * </blockquote>
307      * </p>
308      * <p>
309      * <b>Granting all access:</b> the keyword "all" grants permission to all protocols.
310      * </p>
311      *
312      * <p>
313      * <b>System Property:</b> The value of this property can be set or overridden by
314      * system property {@code javax.xml.accessExternalSchema}
315      * </p>
316      *
317      * <p>
318      * <b>${JAVA_HOME}/lib/jaxp.properties:</b> This configuration file is in standard
319      * java.util.Properties format. If the file exists and the system property is specified,
320      * its value will be used to override the default of the property.
321      *
322      * @since 1.7
323      * </p>
324      */
325      public static final String ACCESS_EXTERNAL_SCHEMA = "http://javax.xml.XMLConstants/property
      /accessExternalSchema";
326
327      /**
328      * <p>Property: accessExternalStylesheet</p>
329      *
330      * <p>
331      * Restrict access to the protocols specified for external references set by the
332      * stylesheet processing instruction, Import and Include element, and document function.
333      * If access is denied due to the restriction of this property, a runtime exception
334      * that is specific to the context is thrown. In the case of constructing new
335      * {@link javax.xml.transform.Transformer} for example,
336      * {@link javax.xml.transform.TransformerConfigurationException}
337      * will be thrown by the {@link javax.xml.transform.TransformerFactory}.
338      * </p>
339      * <p>
340      * <b>Value:</b> a list of protocols separated by comma. A protocol is the scheme portion
      * of a
341      * {@link java.net.URI}, or in the case of the JAR protocol, "jar" plus the scheme portion
342      * separated by colon.
343      * A scheme is defined as:

```



```

344      *
345      * <blockquote>
346      * scheme = alpha *( alpha | digit | "+" | "-" | "." )<br>
347      * where alpha = a-z and A-Z.<br><br>
348      *
349      * And the JAR protocol:<br>
350      *
351      * jar[:scheme]<br><br>
352      *
353      * Protocols including the keyword "jar" are case-insensitive. Any whitespaces as defined
354      * by
355      * {@link java.lang.Character#isSpaceChar } in the value will be ignored.
356      * Examples of protocols are file, http, jar:file.
357      * </blockquote>
358      *</p>
359      *
360      *<p>
361      * <b>Default value:</b> The default value is implementation specific and therefore not
362      * specified.
363      * The following options are provided for consideration:
364      * <blockquote>
365      * <ul>
366      * <li>an empty string to deny all access to external references;</li>
367      * <li>a specific protocol, such as file, to give permission to only the protocol;</li>
368      * <li>the keyword "all" to grant permission to all protocols.</li>
369      * </ul><br>
370      * When FEATURE_SECURE_PROCESSING is enabled, it is recommended that implementations
371      * restrict external connections by default, though this may cause problems for
372      * applications
373      * that process XML/XSD/XSL with external references.
374      * </blockquote>
375      * </p>
376      * <p>
377      * <b>Granting all access:</b> the keyword "all" grants permission to all protocols.
378      * </p>
379      *
380      * <p>
381      * <b>System Property:</b> The value of this property can be set or overridden by
382      * system property {@code javax.xml.accessExternalStylesheet}
383      * </p>
384      *
385      * <p>
386      * <b>${JAVA_HOME}/lib/jaxp.properties:</b> This configuration file is in standard
387      * java.util.Properties format. If the file exists and the system property is specified,
388      * its value will be used to override the default of the property.
389      *
390      * @since 1.7
391      */
392      public static final String ACCESS_EXTERNAL_STYLESHEET = "http://javax.xml.XMLConstants/
    property/accessExternalStylesheet";

```

## 2 Xml Bind (javax.xml.bind package)

### 2.1 Binder.java

Secuencia 3: javax/xml/bind/Binder.java

```
1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import org.w3c.dom.Node;
29
30 import javax.xml.validation.Schema;
31
32 /**
33  * Enable synchronization between XML infoset nodes and JAXB objects
34  * representing same XML document.
35  *
36  * <p>
37  * An instance of this class maintains the association between XML nodes of
38  * an infoset preserving view and a JAXB representation of an XML document.
39  * Navigation between the two views is provided by the methods
40  * {@link #getXMLNode(Object)} and {@link #getJAXBNode(Object)}.
41  *
42  * <p>
43  * Modifications can be made to either the infoset preserving view or the
44  * JAXB representation of the document while the other view remains
45  * unmodified. The binder is able to synchronize the changes made in the
46  * modified view back into the other view using the appropriate
47  * Binder update methods, {@link #updateXML(Object, Object)} or
```

```

48 * {@link #updateJAXB(Object)}.
49 *
50 * <p>
51 * A typical usage scenario is the following:
52 * <ul>
53 *   <li>load XML document into an XML infoset representation</li>
54 *   <li>{@link #unmarshal(Object)} XML infoset view to JAXB view.
55 *     (Note to conserve resources, it is possible to only unmarshal a
56 *     subtree of the XML infoset view to the JAXB view.)</li>
57 *   <li>application access/updates JAXB view of XML document.</li>
58 *   <li>{@link #updateXML(Object)} synchronizes modifications to JAXB view
59 *     back into the XML infoset view. Update operation preserves as
60 *     much of original XML infoset as possible (i.e. comments, PI, ...)</li>
61 * </ul>
62 *
63 * <p>
64 * A Binder instance is created using the factory method
65 * {@link JAXBContext#createBinder()} or {@link JAXBContext#createBinder(Class)}.
66 *
67 * <p>
68 * The template parameter, XmlNode, is the
69 * root interface/class for the XML infoset preserving representation.
70 * A Binder implementation is required to minimally support
71 * an XmlNode value of org.w3c.dom.Node.class.
72 * A Binder implementation can support alternative XML infoset
73 * preserving representations.
74 *
75 * @author
76 *   Kohsuke Kawaguchi (kohsuke.kawaguchi@sun.com)
77 *   Joseph Fialli
78 *
79 * @since JAXB 2.0
80 */
81 public abstract class Binder<XmlNode> {
82     /**
83      * Unmarshal XML infoset view to a JAXB object tree.
84      *
85      * <p>
86      * This method is similar to {@link Unmarshaller#unmarshal(Node)}
87      * with the addition of maintaining the association between XML nodes
88      * and the produced JAXB objects, enabling future update operations,
89      * {@link #updateXML(Object, Object)} or {@link #updateJAXB(Object)}.
90      *
91      * <p>
92      * When {@link #getSchema()} is non-null, xmlNode
93      * and its descendants is validated during this operation.
94      *
95      * <p>
96      * This method throws {@link UnmarshalException} when the Binder's
97      * {@link JAXBContext} does not have a mapping for the XML element name
98      * or the type, specifiable via <tt>xsi:type</tt>, of <tt>xmlNode</tt>
99      * to a JAXB mapped class. The method {@link #unmarshal(Object, Class)}
100      * enables an application to specify the JAXB mapped class that

```

```

101     * the <tt>xmlNode</tt> should be mapped to.
102     *
103     * @param xmlNode
104     *     the document/element to unmarshal XML data from.
105     *
106     * @return
107     *     the newly created root object of the JAXB object tree.
108     *
109     * @throws JAXBException
110     *     If any unexpected errors occur while unmarshalling
111     * @throws UnmarshalException
112     *     If the {@link ValidationEventHandler ValidationEventHandler}
113     *     returns false from its <tt>handleEvent</tt> method or the
114     *     <tt>Binder</tt> is unable to perform the XML to Java
115     *     binding.
116     * @throws IllegalArgumentException
117     *     If the node parameter is null
118     */
119     public abstract Object unmarshal( XmlNode xmlNode ) throws JAXBException;
120
121     /**
122     * Unmarshal XML root element by provided <tt>declaredType</tt>
123     * to a JAXB object tree.
124     *
125     * <p>
126     * Implements <a href="Unmarshaller.html#unmarshalByDeclaredType">Unmarshal by Declared Type</a>
127     *
128     * <p>
129     * This method is similar to {@link Unmarshaller#unmarshal(Node, Class)}
130     * with the addition of maintaining the association between XML nodes
131     * and the produced JAXB objects, enabling future update operations,
132     * {@link #updateXML(Object, Object)} or {@link #updateJAXB(Object)}.
133     *
134     * <p>
135     * When {@link #getSchema()} is non-null, <code>xmlNode</code>
136     * and its descendants is validated during this operation.
137     *
138     * @param xmlNode
139     *     the document/element to unmarshal XML data from.
140     * @param declaredType
141     *     appropriate JAXB mapped class to hold <tt>node</tt>'s XML data.
142     *
143     * @return
144     *     <a href="JAXBElement.html">JAXB Element</a> representation
145     *     of <tt>node</tt>
146     *
147     * @throws JAXBException
148     *     If any unexpected errors occur while unmarshalling
149     * @throws UnmarshalException
150     *     If the {@link ValidationEventHandler ValidationEventHandler}
151     *     returns false from its <tt>handleEvent</tt> method or the
152     *     <tt>Binder</tt> is unable to perform the XML to Java

```

```

153     *      binding.
154     * @throws IllegalArgumentException
155     *      If any of the input parameters are null
156     * @since JAXB2.0
157     */
158     public abstract <T> JAXBElement<T>
159         unmarshal( XmlNode xmlNode, Class<T> declaredType )
160         throws JAXBException;
161
162     /**
163     * Marshal a JAXB object tree to a new XML document.
164     *
165     * <p>
166     * This method is similar to {@link Marshaller#marshal(Object, Node)}
167     * with the addition of maintaining the association between JAXB objects
168     * and the produced XML nodes,
169     * enabling future update operations such as
170     * {@link #updateXML(Object, Object)} or {@link #updateJAXB(Object)}.
171     *
172     * <p>
173     * When {@link #getSchema()} is non-null, the marshalled
174     * xml content is validated during this operation.
175     *
176     * @param jaxbObject
177     *      The content tree to be marshalled.
178     * @param xmlNode
179     *      The parameter must be a Node that accepts children.
180     *
181     * @throws JAXBException
182     *      If any unexpected problem occurs during the marshalling.
183     * @throws MarshalException
184     *      If the {@link ValidationEventHandler ValidationEventHandler}
185     *      returns false from its <tt>handleEvent</tt> method or the
186     *      <tt>Binder</tt> is unable to marshal <tt>jaxbObject</tt> (or any
187     *      object reachable from <tt>jaxbObject</tt>).
188     *
189     * @throws IllegalArgumentException
190     *      If any of the method parameters are null
191     */
192     public abstract void marshal( Object jaxbObject, XmlNode xmlNode ) throws JAXBException;
193
194     /**
195     * Gets the XML element associated with the given JAXB object.
196     *
197     * <p>
198     * Once a JAXB object tree is associated with an XML fragment,
199     * this method enables navigation between the two trees.
200     *
201     * <p>
202     * An association between an XML element and a JAXB object is
203     * established by the bind methods and the update methods.
204     * Note that this association is partial; not all XML elements
205     * have associated JAXB objects, and not all JAXB objects have

```

```

206     * associated XML elements.
207     *
208     * @param jaxbObject An instance that is reachable from a prior
209     *                   call to a bind or update method that returned
210     *                   a JAXB object tree.
211     *
212     * @return
213     *     null if the specified JAXB object is not known to this
214     *     {@link Binder}, or if it is not associated with an
215     *     XML element.
216     *
217     * @throws IllegalArgumentException
218     *     If the jaxbObject parameter is null
219     */
220 public abstract XmlNode getXMLNode( Object jaxbObject );
221
222 /**
223  * Gets the JAXB object associated with the given XML element.
224  *
225  * <p>
226  * Once a JAXB object tree is associated with an XML fragment,
227  * this method enables navigation between the two trees.
228  *
229  * <p>
230  * An association between an XML element and a JAXB object is
231  * established by the unmarshal, marshal and update methods.
232  * Note that this association is partial; not all XML elements
233  * have associated JAXB objects, and not all JAXB objects have
234  * associated XML elements.
235  *
236  * @return
237  *     null if the specified XML node is not known to this
238  *     {@link Binder}, or if it is not associated with a
239  *     JAXB object.
240  *
241  * @throws IllegalArgumentException
242  *     If the node parameter is null
243  */
244 public abstract Object getJAXBNode( XmlNode xmlNode );
245
246 /**
247  * Takes an JAXB object and updates
248  * its associated XML node and its descendants.
249  *
250  * <p>
251  * This is a convenience method of:
252  * <pre>
253  * updateXML( jaxbObject, getXMLNode(jaxbObject));
254  * </pre>
255  *
256  * @throws JAXBException
257  *     If any unexpected problem occurs updating corresponding XML content.
258  * @throws IllegalArgumentException

```

```

259     *      If the jaxbObject parameter is null
260     */
261     public abstract XmlNode updateXML( Object jaxbObject ) throws JAXBException;
262
263     /**
264     * Changes in JAXB object tree are updated in its associated XML parse tree.
265     *
266     * <p>
267     * This operation can be thought of as an "in-place" marshalling.
268     * The difference is that instead of creating a whole new XML tree,
269     * this operation updates an existing tree while trying to preserve
270     * the XML as much as possible.
271     *
272     * <p>
273     * For example, unknown elements/attributes in XML that were not bound
274     * to JAXB will be left untouched (whereas a marshalling operation
275     * would create a new tree that doesn't contain any of those.)
276     *
277     * <p>
278     * As a side-effect, this operation updates the association between
279     * XML nodes and JAXB objects.
280     *
281     * @param jaxbObject root of potentially modified JAXB object tree
282     * @param xmlNode     root of update target XML parse tree
283     *
284     * @return
285     *      Returns the updated XML node. Typically, this is the same
286     *      node you passed in as <i>xmlNode</i>, but it maybe
287     *      a different object, for example when the tag name of the object
288     *      has changed.
289     *
290     * @throws JAXBException
291     *      If any unexpected problem occurs updating corresponding XML content.
292     * @throws IllegalArgumentException
293     *      If any of the input parameters are null
294     */
295     public abstract XmlNode updateXML( Object jaxbObject, XmlNode xmlNode ) throws JAXBException;
296
297     /**
298     * Takes an XML node and updates its associated JAXB object and its descendants.
299     *
300     * <p>
301     * This operation can be thought of as an "in-place" unmarshalling.
302     * The difference is that instead of creating a whole new JAXB tree,
303     * this operation updates an existing tree, reusing as much JAXB objects
304     * as possible.
305     *
306     * <p>
307     * As a side-effect, this operation updates the association between
308     * XML nodes and JAXB objects.
309     *
310     * @return
311     *      Returns the updated JAXB object. Typically, this is the same

```

```

312     *      object that was returned from earlier
313     *      {@link #marshal(Object,Object)} or
314     *      {@link #updateJAXB(Object)} method invocation,
315     *      but it maybe
316     *      a different object, for example when the name of the XML
317     *      element has changed.
318     *
319     * @throws JAXBException
320     *      If any unexpected problem occurs updating corresponding JAXB mapped content.
321     * @throws IllegalArgumentException
322     *      If node parameter is null
323     */
324 public abstract Object updateJAXB( XmlNode xmlNode ) throws JAXBException;
325
326
327 /**
328  * Specifies whether marshal, unmarshal and update methods
329  * performs validation on their XML content.
330  *
331  * @param schema set to null to disable validation.
332  *
333  * @see Unmarshaller#setSchema(Schema)
334  */
335 public abstract void setSchema( Schema schema );
336
337 /**
338  * Gets the last {@link Schema} object (including null) set by the
339  * {@link #setSchema(Schema)} method.
340  *
341  * @return the Schema object for validation or null if not present
342  */
343 public abstract Schema getSchema();
344
345 /**
346  * Allow an application to register a <tt>ValidationEventHandler</tt>.
347  * <p>
348  * The <tt>ValidationEventHandler</tt> will be called by the JAXB Provider
349  * if any validation errors are encountered during calls to any of the
350  * Binder unmarshal, marshal and update methods.
351  *
352  * <p>
353  * Calling this method with a null parameter will cause the Binder
354  * to revert back to the default default event handler.
355  *
356  * @param handler the validation event handler
357  * @throws JAXBException if an error was encountered while setting the
358  *      event handler
359  */
360 public abstract void setEventHandler( ValidationEventHandler handler ) throws JAXBException;
361
362 /**
363  * Return the current event handler or the default event handler if one
364  * hasn't been set.

```



```

365     *
366     * @return the current ValidationEventHandler or the default event handler
367     *         if it hasn't been set
368     * @throws JAXBException if an error was encountered while getting the
369     *         current event handler
370     */
371     public abstract ValidationEventHandler getEventHandler() throws JAXBException;
372
373     /**
374     *
375     * Set the particular property in the underlying implementation of
376     * <tt>Binder</tt>. This method can only be used to set one of
377     * the standard JAXB defined unmarshal/marshal properties
378     * or a provider specific property for binder, unmarshal or marshal.
379     * Attempting to set an undefined property will result in
380     * a PropertyException being thrown. See
381     * <a href="Unmarshaller.html#supportedProps">Supported Unmarshal Properties</a>
382     * and
383     * <a href="Marshaller.html#supportedProps">Supported Marshal Properties</a>.
384     *
385     * @param name the name of the property to be set. This value can either
386     *             be specified using one of the constant fields or a user
387     *             supplied string.
388     * @param value the value of the property to be set
389     *
390     * @throws PropertyException when there is an error processing the given
391     *             property or value
392     * @throws IllegalArgumentException
393     *             If the name parameter is null
394     */
395     abstract public void setProperty( String name, Object value ) throws PropertyException;
396
397
398     /**
399     * Get the particular property in the underlying implementation of
400     * <tt>Binder</tt>. This method can only
401     * be used to get one of
402     * the standard JAXB defined unmarshal/marshal properties
403     * or a provider specific property for binder, unmarshal or marshal.
404     * Attempting to get an undefined property will result in
405     * a PropertyException being thrown. See
406     * <a href="Unmarshaller.html#supportedProps">Supported Unmarshal Properties</a>
407     * and
408     * <a href="Marshaller.html#supportedProps">Supported Marshal Properties</a>.
409     *
410     * @param name the name of the property to retrieve
411     * @return the value of the requested property
412     *
413     * @throws PropertyException
414     *             when there is an error retrieving the given property or value
415     *             property name
416     * @throws IllegalArgumentException
417     *             If the name parameter is null

```

```
418     */
419     abstract public Object getProperty( String name ) throws PropertyException;
420
421 }
```

## 2.2 ContextFinder.java

Secuencia 4: javax/xml/bind/ContextFinder.java

```
1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import java.util.Iterator;
29 import java.io.BufferedReader;
30 import java.io.IOException;
31 import java.io.InputStream;
32 import java.io.InputStreamReader;
33 import java.io.UnsupportedEncodingException;
34 import java.lang.reflect.InvocationTargetException;
35 import java.lang.reflect.Method;
36 import java.net.URL;
37 import java.util.Map;
38 import java.util.Properties;
39 import java.util.StringTokenizer;
40 import java.util.logging.ConsoleHandler;
41 import java.util.logging.Level;
42 import java.util.logging.Logger;
43 import java.security.AccessController;
44
45 import static javax.xml.bind.JAXBContext.JAXB_CONTEXT_FACTORY;
```

```
46
47
48 /**
49  * This class is package private and therefore is not exposed as part of the
50  * JAXB API.
51  *
52  * This code is designed to implement the JAXB 1.0 spec pluggability feature
53  *
54  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
55  * @see JAXBContext
56  */
57 class ContextFinder {
58     private static final Logger logger;
59     static {
60         logger = Logger.getLogger("javax.xml.bind");
61         try {
62             if (AccessController.doPrivileged(new GetPropertyAction("jaxb.debug")) != null) {
63                 // disconnect the logger from a bigger framework (if any)
64                 // and take the matters into our own hands
65                 logger.setUseParentHandlers(false);
66                 logger.setLevel(Level.ALL);
67                 ConsoleHandler handler = new ConsoleHandler();
68                 handler.setLevel(Level.ALL);
69                 logger.addHandler(handler);
70             } else {
71                 // don't change the setting of this logger
72                 // to honor what other frameworks
73                 // have done on configurations.
74             }
75         } catch (Throwable t) {
76             // just to be extra safe. in particular System.getProperty may throw
77             // SecurityException.
78         }
79     }
80
81     /**
82      * If the {@link InvocationTargetException} wraps an exception that shouldn't be wrapped,
83      * throw the wrapped exception.
84      */
85     private static void handleInvocationTargetException(InvocationTargetException x) throws
86         JAXBException {
87         Throwable t = x.getTargetException();
88         if (t != null) {
89             if (t instanceof JAXBException)
90                 // one of our exceptions, just re-throw
91                 throw (JAXBException)t;
92             if (t instanceof RuntimeException)
93                 // avoid wrapping exceptions unnecessarily
94                 throw (RuntimeException)t;
95             if (t instanceof Error)
96                 throw (Error)t;
97         }
98     }
99 }
```

```

98
99
100 /**
101  * Determine if two types (JAXBContext in this case) will generate a ClassCastException.
102  *
103  * For example, (targetType)originalType
104  *
105  * @param originalType
106  *         The Class object of the type being cast
107  * @param targetType
108  *         The Class object of the type that is being cast to
109  * @return JAXBException to be thrown.
110  */
111 private static JAXBException handleClassCastException(Class originalType, Class targetType) {
112     final URL targetTypeURL = which(targetType);
113
114     return new JAXBException(Messages.format(Messages.ILLEGAL_CAST,
115         // we don't care where the impl class is, we want to know where JAXBContext lives
116         // in the impl
117         // class' ClassLoader
118         getClassClassLoader(originalType).getResource("javax/xml/bind/JAXBContext.class"),
119         targetTypeURL));
120 }
121
122 /**
123  * Create an instance of a class using the specified ClassLoader
124  */
125 static JAXBContext newInstance( String contextPath,
126     String className,
127     ClassLoader classLoader,
128     Map properties )
129     throws JAXBException {
130     try {
131         Class spFactory = safeLoadClass(className, classLoader);
132         return newInstance(contextPath, spFactory, classLoader, properties);
133     } catch (ClassNotFoundException x) {
134         throw new JAXBException(
135             Messages.format( Messages.PROVIDER_NOT_FOUND, className ),
136             x);
137     } catch (RuntimeException x) {
138         // avoid wrapping RuntimeException to JAXBException,
139         // because it indicates a bug in this code.
140         throw x;
141     } catch (Exception x) {
142         // can't catch JAXBException because the method is hidden behind
143         // reflection. Root element collisions detected in the call to
144         // createContext() are reported as JAXBExceptions - just re-throw it
145         // some other type of exception - just wrap it
146         throw new JAXBException(
147             Messages.format( Messages.COULD_NOT_INSTANTIATE, className, x ),
148             x);
149     }
150 }

```

```

150
151     static JAXBContext newInstance( String contextPath,
152                                     Class spFactory,
153                                     ClassLoader classLoader,
154                                     Map properties )
155     throws JAXBException
156     {
157         try {
158             /*
159              * javax.xml.bind.context.factory points to a class which has a
160              * static method called 'createContext' that
161              * returns a javax.xml.JAXBContext.
162              */
163
164             Object context = null;
165
166             // first check the method that takes Map as the third parameter.
167             // this is added in 2.0.
168             try {
169                 Method m = spFactory.getMethod("createContext",String.class,ClassLoader.class,Map.
170                                                 class);
171                 // any failure in invoking this method would be considered fatal
172                 context = m.invoke(null,contextPath,classLoader,properties);
173             } catch (NoSuchMethodException e) {
174                 // it's not an error for the provider not to have this method.
175             }
176
177             if(context==null) {
178                 // try the old method that doesn't take properties. compatible with 1.0.
179                 // it is an error for an implementation not to have both forms of the createContext
180                 // method.
181                 Method m = spFactory.getMethod("createContext",String.class,ClassLoader.class);
182                 // any failure in invoking this method would be considered fatal
183                 context = m.invoke(null,contextPath,classLoader);
184             }
185
186             if(!(context instanceof JAXBContext)) {
187                 // the cast would fail, so generate an exception with a nice message
188                 throw handleClassCastException(context.getClass(), JAXBContext.class);
189             }
190             return (JAXBContext)context;
191         } catch (InvocationTargetException x) {
192             handleInvocationTargetException(x);
193             // for other exceptions, wrap the internal target exception
194             // with a JAXBException
195             Throwable e = x;
196             if(x.getTargetException()!=null)
197                 e = x.getTargetException();
198
199             throw new JAXBException( Messages.format( Messages.COULD_NOT_INSTANTIATE, spFactory, e
200                                                         ), e );
201         } catch (RuntimeException x) {
202             // avoid wrapping RuntimeException to JAXBException,

```

```

200         // because it indicates a bug in this code.
201         throw x;
202     } catch (Exception x) {
203         // can't catch JAXBException because the method is hidden behind
204         // reflection. Root element collisions detected in the call to
205         // createContext() are reported as JAXBExceptions - just re-throw it
206         // some other type of exception - just wrap it
207         throw new JAXBException(
208             Messages.format( Messages.COULD_NOT_INSTANTIATE, spFactory, x ),
209             x);
210     }
211 }
212
213
214 /**
215  * Create an instance of a class using the thread context ClassLoader
216  */
217 static JAXBContext newInstance(
218     Class[] classes,
219     Map properties,
220     String className) throws JAXBException {
221     ClassLoader cl = getContextClassLoader();
222     Class spi;
223     try {
224         spi = safeLoadClass(className, cl);
225     } catch (ClassNotFoundException e) {
226         throw new JAXBException(e);
227     }
228
229     if(logger.isLoggable(Level.FINE)) {
230         // extra check to avoid costly which operation if not logged
231         logger.log(Level.FINE, "loaded {0} from {1}", new Object[]{className, which(spi)});
232     }
233
234     return newInstance(classes, properties, spi);
235 }
236
237 static JAXBContext newInstance(Class[] classes,
238     Map properties,
239     Class spFactory) throws JAXBException {
240     Method m;
241     try {
242         m = spFactory.getMethod("createContext", Class[].class, Map.class);
243     } catch (NoSuchMethodException e) {
244         throw new JAXBException(e);
245     }
246     try {
247         Object context = m.invoke(null, classes, properties);
248         if(!(context instanceof JAXBContext)) {
249             // the cast would fail, so generate an exception with a nice message
250             throw handleClassCastException(context.getClass(), JAXBContext.class);
251         }
252         return (JAXBContext)context;

```

```
253     } catch (IllegalAccessException e) {
254         throw new JAXBException(e);
255     } catch (InvocationTargetException e) {
256         handleInvocationTargetException(e);
257
258         Throwable x = e;
259         if (e.getTargetException() != null)
260             x = e.getTargetException();
261
262         throw new JAXBException(x);
263     }
264 }
265
266 static JAXBContext find(String factoryId, String contextPath, ClassLoader classLoader, Map
    properties ) throws JAXBException {
267
268     // TODO: do we want/need another layer of searching in $java.home/lib/jaxb.properties like
269     // JAXP?
270
271     final String jaxbContextFQCN = JAXBContext.class.getName();
272
273     // search context path for jaxb.properties first
274     StringBuilder propFileName;
275     StringTokenizer packages = new StringTokenizer( contextPath, ":" );
276     String factoryClassName;
277
278     if(!packages.hasMoreTokens())
279         // no context is specified
280         throw new JAXBException(Messages.format(Messages.NO_PACKAGE_IN_CONTEXTPATH));
281
282     logger.fine("Searching jaxb.properties");
283
284     while( packages.hasMoreTokens() ) {
285         String packageName = packages.nextToken(":").replace('.', '/');
286         // com.acme.foo -> com/acme/foo/jaxb.properties
287         propFileName = new StringBuilder().append(packageName).append("/jaxb.properties");
288
289         Properties props = loadJAXBProperties( classLoader, propFileName.toString() );
290         if (props != null) {
291             if (props.containsKey(factoryId)) {
292                 factoryClassName = props.getProperty(factoryId);
293                 return newInstance( contextPath, factoryClassName, classLoader, properties );
294             } else {
295                 throw new JAXBException(Messages.format(Messages.MISSING_PROPERTY, packageName,
296                     factoryId));
297             }
298         }
299     }
300
301     logger.fine("Searching the system property");
302
303     // search for a system property second (javax.xml.bind.JAXBContext)
```

```
303     factoryClassName = AccessController.doPrivileged(new GetPropertyAction(JAXBContext.  
304         JAXB_CONTEXT_FACTORY));  
305     if( factoryClassName != null ) {  
306         return newInstance( contextPath, factoryClassName, classLoader, properties );  
307     } else { // leave this here to assure compatibility  
308         factoryClassName = AccessController.doPrivileged(new GetPropertyAction(jaxbContextFQCN)  
309             );  
310         if( factoryClassName != null ) {  
311             return newInstance( contextPath, factoryClassName, classLoader, properties );  
312         }  
313     }  
314  
315     // OSGi search  
316     Class jaxbContext = lookupJaxbContextUsingOsgiServiceLoader();  
317     if (jaxbContext != null) {  
318         logger.fine("OSGi environment detected");  
319         return newInstance(contextPath, jaxbContext, classLoader, properties);  
320     }  
321  
322     logger.fine("Searching META-INF/services");  
323     // search META-INF services next  
324     BufferedReader r = null;  
325     try {  
326         final StringBuilder resource = new StringBuilder().append("META-INF/services/").append(  
327             jaxbContextFQCN);  
328         final InputStream resourceStream =  
329             classLoader.getResourceAsStream(resource.toString());  
330  
331         if (resourceStream != null) {  
332             r = new BufferedReader(new InputStreamReader(resourceStream, "UTF-8"));  
333             factoryClassName = r.readLine();  
334             if (factoryClassName != null) {  
335                 factoryClassName = factoryClassName.trim();  
336             }  
337             r.close();  
338             return newInstance(contextPath, factoryClassName, classLoader, properties);  
339         } else {  
340             logger.log(Level.FINE, "Unable to load:{0}", resource.toString());  
341         }  
342     } catch (UnsupportedEncodingException e) {  
343         // should never happen  
344         throw new JAXBException(e);  
345     } catch (IOException e) {  
346         throw new JAXBException(e);  
347     } finally {  
348         try {  
349             if (r != null) {  
350                 r.close();  
351             }  
352         } catch (IOException ex) {  
353             Logger.getLogger(ContextFinder.class.getName()).log(Level.SEVERE, null, ex);  
354         }  
355     }
```



```

353
354     // else no provider found
355     logger.fine("Trying to create the platform default provider");
356     return newInstance(contextPath, PLATFORM_DEFAULT_FACTORY_CLASS, classLoader, properties);
357 }
358
359 static JAXBContext find( Class[] classes, Map properties ) throws JAXBException {
360
361     final String jaxbContextFQCN = JAXBContext.class.getName();
362     String factoryClassName;
363
364     // search for jaxb.properties in the class loader of each class first
365     for (final Class c : classes) {
366         // this classloader is used only to load jaxb.properties, so doing this should be safe.
367         ClassLoader classLoader = getClassClassLoader(c);
368         Package pkg = c.getPackage();
369         if(pkg==null)
370             continue; // this is possible for primitives, arrays, and classes that are
371                        // loaded by poorly implemented ClassLoaders
372         String packageName = pkg.getName().replace('.', '/');
373
374         // TODO: do we want to optimize away searching the same package? org.Foo, org.Bar, com
375                // .Baz
376         // classes from the same package might come from different class loads, so it
377         // might be a bad idea
378
379         // TODO: it's easier to look things up from the class
380         // c.getResourceAsStream("jaxb.properties");
381
382         // build the resource name and use the property loader code
383         String resourceName = packageName+"/jaxb.properties";
384         logger.log(Level.FINE, "Trying to locate {0}", resourceName);
385         Properties props = loadJAXBProperties(classLoader, resourceName);
386         if (props == null) {
387             logger.fine(" not found");
388         } else {
389             logger.fine(" found");
390             if (props.containsKey(JAXB_CONTEXT_FACTORY)) {
391                 // trim() seems redundant, but adding to satisfy customer complaint
392                 factoryClassName = props.getProperty(JAXB_CONTEXT_FACTORY).trim();
393                 return newInstance(classes, properties, factoryClassName);
394             } else {
395                 throw new JAXBException(Messages.format(Messages.MISSING_PROPERTY, packageName,
396                    JAXB_CONTEXT_FACTORY));
397             }
398         }
399     }
400
401     // search for a system property second (javax.xml.bind.JAXBContext)
402     logger.log(Level.FINE, "Checking system property {0}", JAXBContext.JAXB_CONTEXT_FACTORY);
403     factoryClassName = AccessController.doPrivileged(new GetPropertyAction(JAXBContext.
404         JAXB_CONTEXT_FACTORY));
405     if (factoryClassName != null) {

```

```

401     logger.log(Level.FINE, " found {0}", factoryClassName);
402     return newInstance( classes, properties, factoryClassName );
403 } else { // leave it here for compatibility reasons
404     logger.fine(" not found");
405     logger.log(Level.FINE, "Checking system property {0}", jaxbContextFQCN);
406     factoryClassName = AccessController.doPrivileged(new GetPropertyAction(jaxbContextFQCN)
407         );
408     if (factoryClassName != null) {
409         logger.log(Level.FINE, " found {0}", factoryClassName);
410         return newInstance( classes, properties, factoryClassName );
411     } else {
412         logger.fine(" not found");
413     }
414 }
415
416 // OSGi search
417 Class jaxbContext = lookupJaxbContextUsingOsgiServiceLoader();
418 if (jaxbContext != null) {
419     logger.fine("OSGi environment detected");
420     return newInstance(classes, properties, jaxbContext);
421 }
422
423 // search META-INF services next
424 logger.fine("Checking META-INF/services");
425 BufferedReader r = null;
426 try {
427     final String resource = new StringBuilder("META-INF/services/").append(jaxbContextFQCN)
428         .toString();
429     ClassLoader classLoader = getContextClassLoader();
430     URL resourceURL;
431     if(classLoader==null)
432         resourceURL = ClassLoader.getResource(resource);
433     else
434         resourceURL = classLoader.getResource(resource);
435
436     if (resourceURL != null) {
437         logger.log(Level.FINE, "Reading {0}", resourceURL);
438         r = new BufferedReader(new InputStreamReader(resourceURL.openStream(), "UTF-8"));
439         factoryClassName = r.readLine();
440         if (factoryClassName != null) {
441             factoryClassName = factoryClassName.trim();
442         }
443         return newInstance(classes, properties, factoryClassName);
444     } else {
445         logger.log(Level.FINE, "Unable to find: {0}", resource);
446     }
447 } catch (UnsupportedEncodingException e) {
448     // should never happen
449     throw new JAXBException(e);
450 } catch (IOException e) {
451     throw new JAXBException(e);
452 } finally {
453     if (r != null) {

```

```

452         try {
453             r.close();
454         } catch (IOException ex) {
455             logger.log(Level.FINE, "Unable to close stream", ex);
456         }
457     }
458 }
459
460 // else no provider found
461 logger.fine("Trying to create the platform default provider");
462 return newInstance(classes, properties, PLATFORM_DEFAULT_FACTORY_CLASS);
463 }
464
465 private static Class lookupJaxbContextUsingOsgiServiceLoader() {
466     try {
467         // Use reflection to avoid having any dependency on ServiceLoader class
468         Class target = Class.forName("com.sun.org.glassfish.hk2.osgiresourcelocator.
            ServiceLoader");
469         Method m = target.getMethod("lookupProviderClasses", Class.class);
470         Iterator iter = ((Iterable) m.invoke(null, JAXBContext.class)).iterator();
471         return iter.hasNext() ? (Class)iter.next() : null;
472     } catch (Exception e) {
473         logger.log(Level.FINE, "Unable to find from OSGi: javax.xml.bind.JAXBContext");
474         return null;
475     }
476 }
477
478 private static Properties loadJAXBProperties( ClassLoader classLoader,
479                                             String propFileName )
480     throws JAXBException {
481
482     Properties props = null;
483
484     try {
485         URL url;
486         if(classLoader==null)
487             url = ClassLoader.getResource(propFileName);
488         else
489             url = classLoader.getResource( propFileName );
490
491         if( url != null ) {
492             logger.log(Level.FINE, "loading props from {0}", url);
493             props = new Properties();
494             InputStream is = url.openStream();
495             props.load( is );
496             is.close();
497         }
498     } catch( IOException ioe ) {
499         logger.log(Level.FINE, "Unable to load "+propFileName,ioe);
500         throw new JAXBException( ioe.toString(), ioe );
501     }
502
503     return props;

```

```

504     }
505
506
507     /**
508      * Search the given ClassLoader for an instance of the specified class and
509      * return a string representation of the URL that points to the resource.
510      *
511      * @param clazz
512      *         The class to search for
513      * @param loader
514      *         The ClassLoader to search. If this parameter is null, then the
515      *         system class loader will be searched
516      * @return
517      *         the URL for the class or null if it wasn't found
518      */
519     static URL which(Class clazz, ClassLoader loader) {
520
521         String classnameAsResource = clazz.getName().replace('.', '/') + ".class";
522
523         if(loader == null) {
524             loader = getSystemClassLoader();
525         }
526
527         return loader.getResource(classnameAsResource);
528     }
529
530     /**
531      * Get the URL for the Class from it's ClassLoader.
532      *
533      * Convenience method for {@link #which(Class, ClassLoader)}.
534      *
535      * Equivalent to calling: which(clazz, clazz.getClassLoader())
536      *
537      * @param clazz
538      *         The class to search for
539      * @return
540      *         the URL for the class or null if it wasn't found
541      */
542     static URL which(Class clazz) {
543         return which(clazz, getClassClassLoader(clazz));
544     }
545
546     /**
547      * When JAXB is in J2SE, rt.jar has to have a JAXB implementation.
548      * However, rt.jar cannot have META-INF/services/javax.xml.bind.JAXBContext
549      * because if it has, it will take precedence over any file that applications have
550      * in their jar files.
551      *
552      * <p>
553      * When the user bundles his own JAXB implementation, we'd like to use it, and we
554      * want the platform default to be used only when there's no other JAXB provider.
555      *
556      * <p>

```

```
557     * For this reason, we have to hard-code the class name into the API.
558     */
559     private static final String PLATFORM_DEFAULT_FACTORY_CLASS = "com.sun.xml.internal.bind.v2.
        ContextFactory";
560
561     /**
562     * Loads the class, provided that the calling thread has an access to the class being loaded.
563     */
564     private static Class safeLoadClass(String className, ClassLoader classLoader) throws
        ClassNotFoundException {
565         logger.log(Level.FINE, "Trying to load {0}", className);
566         try {
567             // make sure that the current thread has an access to the package of the given name.
568             SecurityManager s = System.getSecurityManager();
569             if (s != null) {
570                 int i = className.lastIndexOf('.');
571                 if (i != -1) {
572                     s.checkPackageAccess(className.substring(0,i));
573                 }
574             }
575
576             if (classLoader == null) {
577                 return Class.forName(className);
578             } else {
579                 return classLoader.loadClass(className);
580             }
581         } catch (SecurityException se) {
582             // anyone can access the platform default factory class without permission
583             if (PLATFORM_DEFAULT_FACTORY_CLASS.equals(className)) {
584                 return Class.forName(className);
585             }
586             throw se;
587         }
588     }
589
590     private static ClassLoader getContextClassLoader() {
591         if (System.getSecurityManager() == null) {
592             return Thread.currentThread().getContextClassLoader();
593         } else {
594             return (ClassLoader) java.security.AccessController.doPrivileged(
595                 new java.security.PrivilegedAction() {
596                     public java.lang.Object run() {
597                         return Thread.currentThread().getContextClassLoader();
598                     }
599                 });
600         }
601     }
602
603     private static ClassLoader getClassClassLoader(final Class c) {
604         if (System.getSecurityManager() == null) {
605             return c.getClassLoader();
606         } else {
607             return (ClassLoader) java.security.AccessController.doPrivileged(
```

```

608         new java.security.PrivilegedAction() {
609             public java.lang.Object run() {
610                 return c.getClassLoader();
611             }
612         });
613     }
614 }
615
616 private static ClassLoader getSystemClassLoader() {
617     if (System.getSecurityManager() == null) {
618         return ClassLoader.getSystemClassLoader();
619     } else {
620         return (ClassLoader) java.security.AccessController.doPrivileged(
621             new java.security.PrivilegedAction() {
622                 public java.lang.Object run() {
623                     return ClassLoader.getSystemClassLoader();
624                 }
625             });
626     }
627 }
628
629 }

```

## 2.3 DataBindingException.java

Secuencia 5: javax/xml/bind/DataBindingException.java

```

1  /*
2  * Copyright (c) 2006, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27

```

```

28 /**
29  * Exception that represents a failure in a JAXB operation.
30  *
31  * <p>
32  * This exception differs from {@link JAXBException} in that
33  * this is an unchecked exception, while <tt>JAXBException</tt>
34  * is a checked exception.
35  *
36  * @see JAXB
37  * @since JAXB2.1
38  */
39 public class DataBindingException extends RuntimeException {
40     public DataBindingException(String message, Throwable cause) {
41         super(message, cause);
42     }
43
44     public DataBindingException(Throwable cause) {
45         super(cause);
46     }
47 }

```

## 2.4 DatatypeConverter.java

Secuencia 6: javax/xml/bind/DatatypeConverter.java

```

1  /**
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import javax.xml.namespace.NamespaceContext;
29

```

```

30 /**
31  * <p>
32  * The javaType binding declaration can be used to customize the binding of
33  * an XML schema datatype to a Java datatype. Customizations can involve
34  * writing a parse and print method for parsing and printing lexical
35  * representations of a XML schema datatype respectively. However, writing
36  * parse and print methods requires knowledge of the lexical representations (
37  * <a href="http://www.w3.org/TR/xmlschema-2/">XML Schema Part2: Datatypes
38  * specification </a>) and hence may be difficult to write.
39  * </p>
40  * <p>
41  * This class makes it easier to write parse and print methods. It defines
42  * static parse and print methods that provide access to a JAXB provider's
43  * implementation of parse and print methods. These methods are invoked by
44  * custom parse and print methods. For example, the binding of xsd:dateTime
45  * to a long can be customized using parse and print methods as follows:
46  * <blockquote>
47  *     <pre>
48  *         // Customized parse method
49  *         public long myParseCal( String dateTimeString ) {
50  *             java.util.Calendar cal = DatatypeConverter.parseDateTime(dateTimeString);
51  *             long longval = convert_calendar_to_long(cal); //application specific
52  *             return longval;
53  *         }
54  *
55  *         // Customized print method
56  *         public String myPrintCal( Long longval ) {
57  *             java.util.Calendar cal = convert_long_to_calendar(longval) ; //application specific
58  *             String dateTimeString = DatatypeConverter.printDateTime(cal);
59  *             return dateTimeString;
60  *         }
61  *     </pre>
62  * </blockquote>
63  * <p>
64  * There is a static parse and print method corresponding to each parse and
65  * print method respectively in the {@link DatatypeConverterInterface
66  * DatatypeConverterInterface}.
67  * <p>
68  * The static methods defined in the class can also be used to specify
69  * a parse or a print method in a javaType binding declaration.
70  * </p>
71  * <p>
72  * JAXB Providers are required to call the
73  * {@link #setDatatypeConverter(DatatypeConverterInterface)
74  * setDatatypeConverter} api at some point before the first marshal or unmarshal
75  * operation (perhaps during the call to JAXBContext.newInstance). This step is
76  * necessary to configure the converter that should be used to perform the
77  * print and parse functionality.
78  * </p>
79  *
80  * <p>
81  * A print method for a XML schema datatype can output any lexical
82  * representation that is valid with respect to the XML schema datatype.

```



```

83  * If an error is encountered during conversion, then an IllegalArgumentException,
84  * or a subclass of IllegalArgumentException must be thrown by the method.
85  * </p>
86  *
87  * @author <ul><li>Sekhar Vajjhala, Sun Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems Inc
      .</li><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Ryan Shoemaker, Sun Microsystems
      Inc.</li></ul>
88  * @see DatatypeConverterInterface
89  * @see ParseConversionEvent
90  * @see PrintConversionEvent
91  * @since JAXB1.0
92  */
93
94  final public class DatatypeConverter {
95
96      // delegate to this instance of DatatypeConverter
97      private static volatile DatatypeConverterInterface theConverter = null;
98
99      private final static JAXBPermission SET_DATATYPE_CONVERTER_PERMISSION =
100          new JAXBPermission("setDatatypeConverter");
101
102      private DatatypeConverter() {
103          // private constructor
104      }
105
106      /**
107       * This method is for JAXB provider use only.
108       * <p>
109       * JAXB Providers are required to call this method at some point before
110       * allowing any of the JAXB client marshal or unmarshal operations to
111       * occur. This is necessary to configure the datatype converter that
112       * should be used to perform the print and parse conversions.
113       *
114       * <p>
115       * Calling this api repeatedly will have no effect - the
116       * DatatypeConverterInterface instance passed into the first invocation is
117       * the one that will be used from then on.
118       *
119       * @param converter an instance of a class that implements the
120       * DatatypeConverterInterface class - this parameter must not be null.
121       * @throws IllegalArgumentException if the parameter is null
122       * @throws SecurityException
123       *      If the {@link SecurityManager} in charge denies the access to
124       *      set the datatype converter.
125       * @see JAXBPermission
126       */
127      public static void setDatatypeConverter( DatatypeConverterInterface converter ) {
128          if( converter == null ) {
129              throw new IllegalArgumentException(
130                  Messages.format( Messages.CONVERTER_MUST_NOT_BE_NULL ) );
131          } else if( theConverter == null ) {
132              SecurityManager sm = System.getSecurityManager();
133              if (sm != null)

```

```

134         sm.checkPermission(SET_DATATYPE_CONVERTER_PERMISSION);
135         theConverter = converter;
136     }
137 }
138
139 private static synchronized void initConverter() {
140     theConverter = new DatatypeConverterImpl();
141 }
142
143 /**
144  * <p>
145  * Convert the lexical XSD string argument into a String value.
146  * @param lexicalXSDString
147  *     A string containing a lexical representation of
148  *     xsd:string.
149  * @return
150  *     A String value represented by the string argument.
151  */
152 public static String parseString( String lexicalXSDString ) {
153     if (theConverter == null) initConverter();
154     return theConverter.parseString( lexicalXSDString );
155 }
156
157 /**
158  * <p>
159  * Convert the string argument into a BigInteger value.
160  * @param lexicalXSDInteger
161  *     A string containing a lexical representation of
162  *     xsd:integer.
163  * @return
164  *     A BigInteger value represented by the string argument.
165  * @throws NumberFormatException <code>lexicalXSDInteger</code> is not a valid string
166  *     representation of a {@link java.math.BigInteger} value.
167  */
168 public static java.math.BigInteger parseInteger( String lexicalXSDInteger ) {
169     if (theConverter == null) initConverter();
170     return theConverter.parseInteger( lexicalXSDInteger );
171 }
172
173 /**
174  * <p>
175  * Convert the string argument into an int value.
176  * @param lexicalXSDInt
177  *     A string containing a lexical representation of
178  *     xsd:int.
179  * @return
180  *     A int value represented by the string argument.
181  * @throws NumberFormatException <code>lexicalXSDInt</code> is not a valid string
182  *     representation of an <code>int</code> value.
183  */
184 public static int parseInt( String lexicalXSDInt ) {
185     if (theConverter == null) initConverter();
186     return theConverter.parseInt( lexicalXSDInt );
187 }

```

```
185     }
186
187     /**
188     * <p>
189     * Converts the string argument into a long value.
190     * @param lexicalXSDLong
191     *     A string containing lexical representation of
192     *     xsd:long.
193     * @return
194     *     A long value represented by the string argument.
195     * @throws NumberFormatException <code>lexicalXSDLong</code> is not a valid string
196     *     representation of a <code>long</code> value.
197     */
198     public static long parseLong( String lexicalXSDLong ) {
199         if (theConverter == null) initConverter();
200         return theConverter.parseLong( lexicalXSDLong );
201     }
202
203     /**
204     * <p>
205     * Converts the string argument into a short value.
206     * @param lexicalXSDShort
207     *     A string containing lexical representation of
208     *     xsd:short.
209     * @return
210     *     A short value represented by the string argument.
211     * @throws NumberFormatException <code>lexicalXSDShort</code> is not a valid string
212     *     representation of a <code>short</code> value.
213     */
214     public static short parseShort( String lexicalXSDShort ) {
215         if (theConverter == null) initConverter();
216         return theConverter.parseShort( lexicalXSDShort );
217     }
218
219     /**
220     * <p>
221     * Converts the string argument into a BigDecimal value.
222     * @param lexicalXSDDecimal
223     *     A string containing lexical representation of
224     *     xsd:decimal.
225     * @return
226     *     A BigDecimal value represented by the string argument.
227     * @throws NumberFormatException <code>lexicalXSDDecimal</code> is not a valid string
228     *     representation of {@link java.math.BigDecimal}.
229     */
230     public static java.math.BigDecimal parseDecimal( String lexicalXSDDecimal ) {
231         if (theConverter == null) initConverter();
232         return theConverter.parseDecimal( lexicalXSDDecimal );
233     }
234
235     /**
236     * <p>
237     * Converts the string argument into a float value.
```

```
235     * @param lexicalXSDFloat
236     *     A string containing lexical representation of
237     *     xsd:float.
238     * @return
239     *     A float value represented by the string argument.
240     * @throws NumberFormatException <code>lexicalXSDFloat</code> is not a valid string
        representation of a <code>float</code> value.
241     */
242     public static float parseFloat( String lexicalXSDFloat ) {
243         if (theConverter == null) initConverter();
244         return theConverter.parseFloat( lexicalXSDFloat );
245     }
246
247     /**
248     * <p>
249     * Converts the string argument into a double value.
250     * @param lexicalXSDDouble
251     *     A string containing lexical representation of
252     *     xsd:double.
253     * @return
254     *     A double value represented by the string argument.
255     * @throws NumberFormatException <code>lexicalXSDDouble</code> is not a valid string
        representation of a <code>double</code> value.
256     */
257     public static double parseDouble( String lexicalXSDDouble ) {
258         if (theConverter == null) initConverter();
259         return theConverter.parseDouble( lexicalXSDDouble );
260     }
261
262     /**
263     * <p>
264     * Converts the string argument into a boolean value.
265     * @param lexicalXSDBoolean
266     *     A string containing lexical representation of
267     *     xsd:boolean.
268     * @return
269     *     A boolean value represented by the string argument.
270     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
        defined in XML Schema Part 2: Datatypes for xsd:boolean.
271     */
272     public static boolean parseBoolean( String lexicalXSDBoolean ) {
273         if (theConverter == null) initConverter();
274         return theConverter.parseBoolean( lexicalXSDBoolean );
275     }
276
277     /**
278     * <p>
279     * Converts the string argument into a byte value.
280     * @param lexicalXSDByte
281     *     A string containing lexical representation of
282     *     xsd:byte.
283     * @return
284     *     A byte value represented by the string argument.
```

```

285     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
        defined in XML Schema Part 2: Datatypes for xsd:byte.
286     */
287     public static byte parseByte( String lexicalXSDByte ) {
288         if (theConverter == null) initConverter();
289         return theConverter.parseByte( lexicalXSDByte );
290     }
291
292     /**
293     * <p>
294     * Converts the string argument into a byte value.
295     *
296     * <p>
297     * String parameter <tt>lexicalXSDQName</tt> must conform to lexical value space specied at
298     * <a href="http://www.w3.org/TR/xmlschema-2/#QName">XML Schema Part 2:Datatypes specification:
        QName</a>
299     *
300     * @param lexicalXSDQName
301     *     A string containing lexical representation of xsd:QName.
302     * @param nsc
303     *     A namespace context for interpreting a prefix within a QName.
304     * @return
305     *     A QName value represented by the string argument.
306     * @throws IllegalArgumentException if string parameter does not conform to XML Schema Part 2
        specification or
307     *     if namespace prefix of <tt>lexicalXSDQName</tt> is not bound to a URI in
        NamespaceContext <tt>nsc</tt>.
308     */
309     public static javax.xml.namespace.QName parseQName( String lexicalXSDQName,
310                                                         NamespaceContext nsc ) {
311         if (theConverter == null) initConverter();
312         return theConverter.parseQName( lexicalXSDQName, nsc );
313     }
314
315     /**
316     * <p>
317     * Converts the string argument into a Calendar value.
318     * @param lexicalXSDDateTime
319     *     A string containing lexical representation of
320     *     xsd:dateTime.
321     * @return
322     *     A Calendar object represented by the string argument.
323     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
        defined in XML Schema Part 2: Datatypes for xsd:dateTime.
324     */
325     public static java.util.Calendar parseDateTime( String lexicalXSDDateTime ) {
326         if (theConverter == null) initConverter();
327         return theConverter.parseDateTime( lexicalXSDDateTime );
328     }
329
330     /**
331     * <p>
332     * Converts the string argument into an array of bytes.

```

```
333     * @param lexicalXSDBase64Binary
334     *     A string containing lexical representation
335     *     of xsd:base64Binary.
336     * @return
337     *     An array of bytes represented by the string argument.
338     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
339     *     defined in XML Schema Part 2: Datatypes for xsd:base64Binary
340     */
341     public static byte[] parseBase64Binary( String lexicalXSDBase64Binary ) {
342         if (theConverter == null) initConverter();
343         return theConverter.parseBase64Binary( lexicalXSDBase64Binary );
344     }
345
346     /**
347     * <p>
348     * Converts the string argument into an array of bytes.
349     * @param lexicalXSDHexBinary
350     *     A string containing lexical representation of
351     *     xsd:hexBinary.
352     * @return
353     *     An array of bytes represented by the string argument.
354     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
355     *     defined in XML Schema Part 2: Datatypes for xsd:hexBinary.
356     */
357     public static byte[] parseHexBinary( String lexicalXSDHexBinary ) {
358         if (theConverter == null) initConverter();
359         return theConverter.parseHexBinary( lexicalXSDHexBinary );
360     }
361
362     /**
363     * <p>
364     * Converts the string argument into a long value.
365     * @param lexicalXSDUnsignedInt
366     *     A string containing lexical representation
367     *     of xsd:unsignedInt.
368     * @return
369     *     A long value represented by the string argument.
370     * @throws NumberFormatException if string parameter can not be parsed into a <tt>long</tt>
371     *     value.
372     */
373     public static long parseUnsignedInt( String lexicalXSDUnsignedInt ) {
374         if (theConverter == null) initConverter();
375         return theConverter.parseUnsignedInt( lexicalXSDUnsignedInt );
376     }
377
378     /**
379     * <p>
380     * Converts the string argument into an int value.
381     * @param lexicalXSDUnsignedShort
382     *     A string containing lexical
383     *     representation of xsd:unsignedShort.
384     * @return
385     *     An int value represented by the string argument.
```

```

383     * @throws NumberFormatException if string parameter can not be parsed into an <tt>int</tt>
      value.
384     */
385     public static int parseUnsignedShort( String lexicalXSDUnsignedShort ) {
386         if (theConverter == null) initConverter();
387         return theConverter.parseUnsignedShort( lexicalXSDUnsignedShort );
388     }
389
390     /**
391     * <p>
392     * Converts the string argument into a Calendar value.
393     * @param lexicalXSDTime
394     *     A string containing lexical representation of
395     *     xsd:time.
396     * @return
397     *     A Calendar value represented by the string argument.
398     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
      defined in XML Schema Part 2: Datatypes for xsd:Time.
399     */
400     public static java.util.Calendar parseTime( String lexicalXSDTime ) {
401         if (theConverter == null) initConverter();
402         return theConverter.parseTime( lexicalXSDTime );
403     }
404
405     /**
406     * <p>
407     * Converts the string argument into a Calendar value.
408     * @param lexicalXSDDate
409     *     A string containing lexical representation of
410     *     xsd:Date.
411     * @return
412     *     A Calendar value represented by the string argument.
413     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
      defined in XML Schema Part 2: Datatypes for xsd:Date.
414     */
415     public static java.util.Calendar parseDate( String lexicalXSDDate ) {
416         if (theConverter == null) initConverter();
417         return theConverter.parseDate( lexicalXSDDate );
418     }
419
420     /**
421     * <p>
422     * Return a string containing the lexical representation of the
423     * simple type.
424     * @param lexicalXSDAnySimpleType
425     *     A string containing lexical
426     *     representation of the simple type.
427     * @return
428     *     A string containing the lexical representation of the
429     *     simple type.
430     */
431     public static String parseAnySimpleType( String lexicalXSDAnySimpleType ) {
432         if (theConverter == null) initConverter();
433         return theConverter.parseAnySimpleType( lexicalXSDAnySimpleType );

```

```
433     }
434     /**
435      * <p>
436      * Converts the string argument into a string.
437      * @param val
438      *     A string value.
439      * @return
440      *     A string containing a lexical representation of xsd:string.
441      */
442     // also indicate the print methods produce a lexical
443     // representation for given Java datatypes.
444
445     public static String printString( String val ) {
446         if (theConverter == null) initConverter();
447         return theConverter.printString( val );
448     }
449
450     /**
451      * <p>
452      * Converts a BigInteger value into a string.
453      * @param val
454      *     A BigInteger value
455      * @return
456      *     A string containing a lexical representation of xsd:integer
457      * @throws IllegalArgumentException <tt>val</tt> is null.
458      */
459     public static String printInteger( java.math.BigInteger val ) {
460         if (theConverter == null) initConverter();
461         return theConverter.printInteger( val );
462     }
463
464     /**
465      * <p>
466      * Converts an int value into a string.
467      * @param val
468      *     An int value
469      * @return
470      *     A string containing a lexical representation of xsd:int
471      */
472     public static String printInt( int val ) {
473         if (theConverter == null) initConverter();
474         return theConverter.printInt( val );
475     }
476
477     /**
478      * <p>
479      * Converts A long value into a string.
480      * @param val
481      *     A long value
482      * @return
483      *     A string containing a lexical representation of xsd:long
484      */
485     public static String printLong( long val ) {
```



```
486         if (theConverter == null) initConverter();
487         return theConverter.printLong( val );
488     }
489
490     /**
491     * <p>
492     * Converts a short value into a string.
493     * @param val
494     *     A short value
495     * @return
496     *     A string containing a lexical representation of xsd:short
497     */
498     public static String printShort( short val ) {
499         if (theConverter == null) initConverter();
500         return theConverter.printShort( val );
501     }
502
503     /**
504     * <p>
505     * Converts a BigDecimal value into a string.
506     * @param val
507     *     A BigDecimal value
508     * @return
509     *     A string containing a lexical representation of xsd:decimal
510     * @throws IllegalArgumentException <tt>val</tt> is null.
511     */
512     public static String printDecimal( java.math.BigDecimal val ) {
513         if (theConverter == null) initConverter();
514         return theConverter.printDecimal( val );
515     }
516
517     /**
518     * <p>
519     * Converts a float value into a string.
520     * @param val
521     *     A float value
522     * @return
523     *     A string containing a lexical representation of xsd:float
524     */
525     public static String printFloat( float val ) {
526         if (theConverter == null) initConverter();
527         return theConverter.printFloat( val );
528     }
529
530     /**
531     * <p>
532     * Converts a double value into a string.
533     * @param val
534     *     A double value
535     * @return
536     *     A string containing a lexical representation of xsd:double
537     */
538     public static String printDouble( double val ) {
```

```
539     if (theConverter == null) initConverter();
540     return theConverter.printDouble( val );
541 }
542
543 /**
544  * <p>
545  * Converts a boolean value into a string.
546  * @param val
547  *     A boolean value
548  * @return
549  *     A string containing a lexical representation of xsd:boolean
550  */
551 public static String printBoolean( boolean val ) {
552     if (theConverter == null) initConverter();
553     return theConverter.printBoolean( val );
554 }
555
556 /**
557  * <p>
558  * Converts a byte value into a string.
559  * @param val
560  *     A byte value
561  * @return
562  *     A string containing a lexical representation of xsd:byte
563  */
564 public static String printByte( byte val ) {
565     if (theConverter == null) initConverter();
566     return theConverter.printByte( val );
567 }
568
569 /**
570  * <p>
571  * Converts a QName instance into a string.
572  * @param val
573  *     A QName value
574  * @param nsc
575  *     A namespace context for interpreting a prefix within a QName.
576  * @return
577  *     A string containing a lexical representation of QName
578  * @throws IllegalArgumentException if <tt>val</tt> is null or
579  *     if <tt>nsc</tt> is non-null or <tt>nsc.getPrefix(nsprefixFromVal)</tt> is null.
580  */
581 public static String printQName( javax.xml.namespace.QName val,
582                                 NamespaceContext nsc ) {
583     if (theConverter == null) initConverter();
584     return theConverter.printQName( val, nsc );
585 }
586
587 /**
588  * <p>
589  * Converts a Calendar value into a string.
590  * @param val
591  *     A Calendar value
```

```
592     * @return
593     *     A string containing a lexical representation of xsd:dateTime
594     * @throws IllegalArgumentException if <tt>val</tt> is null.
595     */
596     public static String printDateTime( java.util.Calendar val ) {
597         if (theConverter == null) initConverter();
598         return theConverter.printDateTime( val );
599     }
600
601     /**
602     * <p>
603     * Converts an array of bytes into a string.
604     * @param val
605     *     An array of bytes
606     * @return
607     *     A string containing a lexical representation of xsd:base64Binary
608     * @throws IllegalArgumentException if <tt>val</tt> is null.
609     */
610     public static String printBase64Binary( byte[] val ) {
611         if (theConverter == null) initConverter();
612         return theConverter.printBase64Binary( val );
613     }
614
615     /**
616     * <p>
617     * Converts an array of bytes into a string.
618     * @param val
619     *     An array of bytes
620     * @return
621     *     A string containing a lexical representation of xsd:hexBinary
622     * @throws IllegalArgumentException if <tt>val</tt> is null.
623     */
624     public static String printHexBinary( byte[] val ) {
625         if (theConverter == null) initConverter();
626         return theConverter.printHexBinary( val );
627     }
628
629     /**
630     * <p>
631     * Converts a long value into a string.
632     * @param val
633     *     A long value
634     * @return
635     *     A string containing a lexical representation of xsd:unsignedInt
636     */
637     public static String printUnsignedInt( long val ) {
638         if (theConverter == null) initConverter();
639         return theConverter.printUnsignedInt( val );
640     }
641
642     /**
643     * <p>
644     * Converts an int value into a string.
```

```
645     * @param val
646     *     An int value
647     * @return
648     *     A string containing a lexical representation of xsd:unsignedShort
649     */
650     public static String printUnsignedShort( int val ) {
651         if (theConverter == null) initConverter();
652         return theConverter.printUnsignedShort( val );
653     }
654
655     /**
656     * <p>
657     * Converts a Calendar value into a string.
658     * @param val
659     *     A Calendar value
660     * @return
661     *     A string containing a lexical representation of xsd:time
662     * @throws IllegalArgumentException if <tt>val</tt> is null.
663     */
664     public static String printTime( java.util.Calendar val ) {
665         if (theConverter == null) initConverter();
666         return theConverter.printTime( val );
667     }
668
669     /**
670     * <p>
671     * Converts a Calendar value into a string.
672     * @param val
673     *     A Calendar value
674     * @return
675     *     A string containing a lexical representation of xsd:date
676     * @throws IllegalArgumentException if <tt>val</tt> is null.
677     */
678     public static String printDate( java.util.Calendar val ) {
679         if (theConverter == null) initConverter();
680         return theConverter.printDate( val );
681     }
682
683     /**
684     * <p>
685     * Converts a string value into a string.
686     * @param val
687     *     A string value
688     * @return
689     *     A string containing a lexical representation of xsd:AnySimpleType
690     */
691     public static String printAnySimpleType( String val ) {
692         if (theConverter == null) initConverter();
693         return theConverter.printAnySimpleType( val );
694     }
695 }
```

## 2.5 DatatypeConverterImpl.java

Secuencia 7: javax/xml/bind/DatatypeConverterImpl.java

```
1  /*
2  * Copyright (c) 2007, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import java.math.BigDecimal;
29 import java.math.BigInteger;
30 import java.util.Calendar;
31 import java.util.GregorianCalendar;
32 import java.util.TimeZone;
33
34 import javax.xml.namespace.QName;
35 import javax.xml.namespace.NamespaceContext;
36 import javax.xml.datatype.DatatypeFactory;
37 import javax.xml.datatype.DatatypeConfigurationException;
38
39 /**
40 * This class is the JAXB RI's default implementation of the
41 * {@link DatatypeConverterInterface}.
42 *
43 * <p>
44 * When client applications specify the use of the static print/parse
45 * methods in {@link DatatypeConverter}, it will delegate
46 * to this class.
47 *
48 * <p>
49 * This class is responsible for whitespace normalization.
50 *
```

```

51  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
52  * @since JAXB2.1
53  */
54  final class DatatypeConverterImpl implements DatatypeConverterInterface {
55
56      /**
57       * To avoid re-creating instances, we cache one instance.
58       */
59      public static final DatatypeConverterInterface theInstance = new DatatypeConverterImpl();
60
61      protected DatatypeConverterImpl() {
62      }
63
64      public String parseString(String lexicalXSDString) {
65          return lexicalXSDString;
66      }
67
68      public BigInteger parseInteger(String lexicalXSDInteger) {
69          return _parseInteger(lexicalXSDInteger);
70      }
71
72      public static BigInteger _parseInteger(CharSequence s) {
73          return new BigInteger(removeOptionalPlus(WhiteSpaceProcessor.trim(s)).toString());
74      }
75
76      public String printInteger(BigInteger val) {
77          return _printInteger(val);
78      }
79
80      public static String _printInteger(BigInteger val) {
81          return val.toString();
82      }
83
84      public int parseInt(String s) {
85          return _parseInt(s);
86      }
87
88      /**
89       * Faster but less robust String->int conversion.
90       *
91       * Note that:
92       * <ol>
93       * <li>XML Schema allows '+', but {@link Integer#valueOf(String)} is not.
94       * <li>XML Schema allows leading and trailing (but not in-between) whitespaces.
95       *     {@link Integer#valueOf(String)} doesn't allow any.
96       * </ol>
97       */
98      public static int _parseInt(CharSequence s) {
99          int len = s.length();
100         int sign = 1;
101
102         int r = 0;
103

```

```
104     for (int i = 0; i < len; i++) {
105         char ch = s.charAt(i);
106         if (WhiteSpaceProcessor.isWhiteSpace(ch)) {
107             // skip whitespace
108         } else if ('0' <= ch && ch <= '9') {
109             r = r * 10 + (ch - '0');
110         } else if (ch == '-') {
111             sign = -1;
112         } else if (ch == '+') {
113             // noop
114         } else {
115             throw new NumberFormatException("Not a number: " + s);
116         }
117     }
118
119     return r * sign;
120 }
121
122 public long parseLong(String lexicalXSLong) {
123     return _parseLong(lexicalXSLong);
124 }
125
126 public static long _parseLong(CharSequence s) {
127     return Long.valueOf(removeOptionalPlus(WhiteSpaceProcessor.trim(s)).toString());
128 }
129
130 public short parseShort(String lexicalXSDDShort) {
131     return _parseShort(lexicalXSDDShort);
132 }
133
134 public static short _parseShort(CharSequence s) {
135     return (short) _parseInt(s);
136 }
137
138 public String printShort(short val) {
139     return _printShort(val);
140 }
141
142 public static String _printShort(short val) {
143     return String.valueOf(val);
144 }
145
146 public BigDecimal parseDecimal(String content) {
147     return _parseDecimal(content);
148 }
149
150 public static BigDecimal _parseDecimal(CharSequence content) {
151     content = WhiteSpaceProcessor.trim(content);
152
153     if (content.length() <= 0) {
154         return null;
155     }
156 }
```

```

157         return new BigDecimal(content.toString());
158
159         // from purely XML Schema perspective,
160         // this implementation has a problem, since
161         // in xs:decimal "1.0" and "1" is equal whereas the above
162         // code will return different values for those two forms.
163         //
164         // the code was originally using com.sun.msv.datatype.xsd.NumberType.load,
165         // but a profiling showed that the process of normalizing "1.0" into "1"
166         // could take non-trivial time.
167         //
168         // also, from the user's point of view, one might be surprised if
169         // 1 (not 1.0) is returned from "1.000"
170     }
171
172     public float parseFloat(String lexicalXSDFloat) {
173         return _parseFloat(lexicalXSDFloat);
174     }
175
176     public static float _parseFloat(CharSequence _val) {
177         String s = WhiteSpaceProcessor.trim(_val).toString();
178         /* Incompatibilities of XML Schema's float "xfloat" and Java's float "jfloat"
179
180         * jfloat.valueOf ignores leading and trailing whitespaces,
181         whereas this is not allowed in xfloat.
182         * jfloat.valueOf allows "float type suffix" (f, F) to be
183         appended after float literal (e.g., 1.52e-2f), whereare
184         this is not the case of xfloat.
185
186         gray zone
187         -----
188         * jfloat allows ".523". And there is no clear statement that mentions
189         this case in xfloat. Although probably this is allowed.
190         *
191         */
192
193         if (s.equals("NaN")) {
194             return Float.NaN;
195         }
196         if (s.equals("INF")) {
197             return Float.POSITIVE_INFINITY;
198         }
199         if (s.equals("-INF")) {
200             return Float.NEGATIVE_INFINITY;
201         }
202
203         if (s.length() == 0
204             || !isDigitOrPeriodOrSign(s.charAt(0))
205             || !isDigitOrPeriodOrSign(s.charAt(s.length() - 1))) {
206             throw new NumberFormatException();
207         }
208
209         // these screening process is necessary due to the wobble of Float.valueOf method

```



```
210     return Float.parseFloat(s);
211 }
212
213 public String printFloat(float v) {
214     return _printFloat(v);
215 }
216
217 public static String _printFloat(float v) {
218     if (Float.isNaN(v)) {
219         return "NaN";
220     }
221     if (v == Float.POSITIVE_INFINITY) {
222         return "INF";
223     }
224     if (v == Float.NEGATIVE_INFINITY) {
225         return "-INF";
226     }
227     return String.valueOf(v);
228 }
229
230 public double parseDouble(String lexicalXSDDouble) {
231     return _parseDouble(lexicalXSDDouble);
232 }
233
234 public static double _parseDouble(CharSequence _val) {
235     String val = WhiteSpaceProcessor.trim(_val).toString();
236
237     if (val.equals("NaN")) {
238         return Double.NaN;
239     }
240     if (val.equals("INF")) {
241         return Double.POSITIVE_INFINITY;
242     }
243     if (val.equals("-INF")) {
244         return Double.NEGATIVE_INFINITY;
245     }
246
247     if (val.length() == 0
248         || !isDigitOrPeriodOrSign(val.charAt(0))
249         || !isDigitOrPeriodOrSign(val.charAt(val.length() - 1))) {
250         throw new NumberFormatException(val);
251     }
252
253     // these screening process is necessary due to the wobble of Float.valueOf method
254     return Double.parseDouble(val);
255 }
256
257 public boolean parseBoolean(String lexicalXSDBoolean) {
258     Boolean b = _parseBoolean(lexicalXSDBoolean);
259     return (b == null) ? false : b.booleanValue();
260 }
261
262
```

```
263 public static Boolean _parseBoolean(CharSequence literal) {
264     if (literal == null) {
265         return null;
266     }
267
268     int i = 0;
269     int len = literal.length();
270     char ch;
271     boolean value = false;
272
273     if (literal.length() <= 0) {
274         return null;
275     }
276
277     do {
278         ch = literal.charAt(i++);
279     } while (WhiteSpaceProcessor.isWhiteSpace(ch) && i < len);
280
281     int strIndex = 0;
282
283     switch (ch) {
284         case '1':
285             value = true;
286             break;
287         case '0':
288             value = false;
289             break;
290         case 't':
291             String strTrue = "true";
292             do {
293                 ch = literal.charAt(i++);
294             } while ((strTrue.charAt(strIndex++) == ch) && i < len && strIndex < 3);
295
296             if (strIndex == 3) {
297                 value = true;
298             } else {
299                 return false;
300             }
301             // throw new IllegalArgumentException("String \"" + literal + "\" is not valid
302             boolean value.");
303             break;
304         case 'f':
305             String strFalse = "false";
306             do {
307                 ch = literal.charAt(i++);
308             } while ((strFalse.charAt(strIndex++) == ch) && i < len && strIndex < 4);
309
310             if (strIndex == 4) {
311                 value = false;
312             } else {
313                 return false;
314             }
```

```
315         }
316 //         throw new IllegalArgumentException("String \"" + literal + "\" is not valid
boolean value.");
317
318         break;
319     }
320
321     if (i < len) {
322         do {
323             ch = literal.charAt(i++);
324         } while (WhiteSpaceProcessor.isWhiteSpace(ch) && i < len);
325     }
326
327     if (i == len) {
328         return value;
329     } else {
330         return null;
331     }
332 //     throw new IllegalArgumentException("String \"" + literal + "\" is not valid boolean
value.");
333 }
334
335 public String printBoolean(boolean val) {
336     return val ? "true" : "false";
337 }
338
339 public static String _printBoolean(boolean val) {
340     return val ? "true" : "false";
341 }
342
343 public byte parseByte(String lexicalXSDByte) {
344     return _parseByte(lexicalXSDByte);
345 }
346
347 public static byte _parseByte(CharSequence literal) {
348     return (byte) _parseInt(literal);
349 }
350
351 public String printByte(byte val) {
352     return _printByte(val);
353 }
354
355 public static String _printByte(byte val) {
356     return String.valueOf(val);
357 }
358
359 public QName parseQName(String lexicalXSDQName, NamespaceContext nsc) {
360     return _parseQName(lexicalXSDQName, nsc);
361 }
362
363 /**
364  * @return null if fails to convert.
365  */
```

```

366 public static QName _parseQName(CharSequence text, NamespaceContext nsc) {
367     int length = text.length();
368
369     // trim whitespace
370     int start = 0;
371     while (start < length && WhiteSpaceProcessor.isWhiteSpace(text.charAt(start))) {
372         start++;
373     }
374
375     int end = length;
376     while (end > start && WhiteSpaceProcessor.isWhiteSpace(text.charAt(end - 1))) {
377         end--;
378     }
379
380     if (end == start) {
381         throw new IllegalArgumentException("input is empty");
382     }
383
384     String uri;
385     String localPart;
386     String prefix;
387
388     // search ':'
389     int idx = start + 1;    // no point in searching the first char. that's not valid.
390     while (idx < end && text.charAt(idx) != ':') {
391         idx++;
392     }
393
394     if (idx == end) {
395         uri = nsc.getNamespaceURI("");
396         localPart = text.subSequence(start, end).toString();
397         prefix = "";
398     } else {
399         // Prefix exists, check everything
400         prefix = text.subSequence(start, idx).toString();
401         localPart = text.subSequence(idx + 1, end).toString();
402         uri = nsc.getNamespaceURI(prefix);
403         // uri can never be null according to javadoc,
404         // but some users reported that there are implementations that return null.
405         if (uri == null || uri.length() == 0) // crap. the NamespaceContext interface is broken
406             .
407         // error: unbound prefix
408         {
409             throw new IllegalArgumentException("prefix " + prefix + " is not bound to a
410                 namespace");
411         }
412     }
413
414     return new QName(uri, localPart, prefix);
415 }
416
417 public Calendar parseDateTime(String lexicalXSDDatetime) {

```

```
417     return _parseDateTime(lexicalXSDDateTime);
418 }
419
420 public static GregorianCalendar _parseDateTime(CharSequence s) {
421     String val = WhiteSpaceProcessor.trim(s).toString();
422     return datatypeFactory.newXMLGregorianCalendar(val).toGregorianCalendar();
423 }
424
425 public String printDateTime(Calendar val) {
426     return _printDateTime(val);
427 }
428
429 public static String _printDateTime(Calendar val) {
430     return CalendarFormatter.doFormat("%Y-%M-%DT%H:%m:%s%Z", val);
431 }
432
433 public byte[] parseBase64Binary(String lexicalXSDBase64Binary) {
434     return _parseBase64Binary(lexicalXSDBase64Binary);
435 }
436
437 public byte[] parseHexBinary(String s) {
438     final int len = s.length();
439
440     // "111" is not a valid hex encoding.
441     if (len % 2 != 0) {
442         throw new IllegalArgumentException("hexBinary needs to be even-length: " + s);
443     }
444
445     byte[] out = new byte[len / 2];
446
447     for (int i = 0; i < len; i += 2) {
448         int h = hexToBin(s.charAt(i));
449         int l = hexToBin(s.charAt(i + 1));
450         if (h == -1 || l == -1) {
451             throw new IllegalArgumentException("contains illegal character for hexBinary: " + s);
452         }
453
454         out[i / 2] = (byte) (h * 16 + l);
455     }
456
457     return out;
458 }
459
460 private static int hexToBin(char ch) {
461     if ('0' <= ch && ch <= '9') {
462         return ch - '0';
463     }
464     if ('A' <= ch && ch <= 'F') {
465         return ch - 'A' + 10;
466     }
467     if ('a' <= ch && ch <= 'f') {
468         return ch - 'a' + 10;
469     }
470 }
```

```
469     }
470     return -1;
471 }
472 private static final char[] hexCode = "0123456789ABCDEF".toCharArray();
473
474 public String printHexBinary(byte[] data) {
475     StringBuilder r = new StringBuilder(data.length * 2);
476     for (byte b : data) {
477         r.append(hexCode[(b >> 4) & 0xF]);
478         r.append(hexCode[(b & 0xF)]);
479     }
480     return r.toString();
481 }
482
483 public long parseUnsignedInt(String lexicalXSDUnsignedInt) {
484     return _parseLong(lexicalXSDUnsignedInt);
485 }
486
487 public String printUnsignedInt(long val) {
488     return _printLong(val);
489 }
490
491 public int parseUnsignedShort(String lexicalXSDUnsignedShort) {
492     return _parseInt(lexicalXSDUnsignedShort);
493 }
494
495 public Calendar parseTime(String lexicalXSDateTime) {
496     return datatypeFactory.newXMLGregorianCalendar(lexicalXSDateTime).toGregorianCalendar();
497 }
498
499 public String printTime(Calendar val) {
500     return CalendarFormatter.doFormat("%h:%m:%s%Z", val);
501 }
502
503 public Calendar parseDate(String lexicalXSDDate) {
504     return datatypeFactory.newXMLGregorianCalendar(lexicalXSDDate).toGregorianCalendar();
505 }
506
507 public String printDate(Calendar val) {
508     return _printDate(val);
509 }
510
511 public static String _printDate(Calendar val) {
512     return CalendarFormatter.doFormat((new StringBuilder("%Y-%M-%D").append("%Z")).toString(),
513         val);
514 }
515
516 public String parseAnySimpleType(String lexicalXSDAnySimpleType) {
517     return lexicalXSDAnySimpleType;
518 }
519 // return (String)SimpleURType.theInstance._createValue( lexicalXSDAnySimpleType, null );
520
521 public String printString(String val) {
```

```
521 //     return StringType.getInstance().convertToLexicalValue( val, null );
522     return val;
523 }
524
525 public String printInt(int val) {
526     return _printInt(val);
527 }
528
529 public static String _printInt(int val) {
530     return String.valueOf(val);
531 }
532
533 public String printLong(long val) {
534     return _printLong(val);
535 }
536
537 public static String _printLong(long val) {
538     return String.valueOf(val);
539 }
540
541 public String printDecimal(BigDecimal val) {
542     return _printDecimal(val);
543 }
544
545 public static String _printDecimal(BigDecimal val) {
546     return val.toString();
547 }
548
549 public String printDouble(double v) {
550     return _printDouble(v);
551 }
552
553 public static String _printDouble(double v) {
554     if (Double.isNaN(v)) {
555         return "NaN";
556     }
557     if (v == Double.POSITIVE_INFINITY) {
558         return "INF";
559     }
560     if (v == Double.NEGATIVE_INFINITY) {
561         return "-INF";
562     }
563     return String.valueOf(v);
564 }
565
566 public String printQName(QName val, NamespaceContext nsc) {
567     return _printQName(val, nsc);
568 }
569
570 public static String _printQName(QName val, NamespaceContext nsc) {
571     // Double-check
572     String qname;
573     String prefix = nsc.getPrefix(val.getNamespaceURI());
```

```
574     String localPart = val.getLocalPart();
575
576     if (prefix == null || prefix.length() == 0) { // be defensive
577         qname = localPart;
578     } else {
579         qname = prefix + ':' + localPart;
580     }
581
582     return qname;
583 }
584
585 public String printBase64Binary(byte[] val) {
586     return _printBase64Binary(val);
587 }
588
589 public String printUnsignedShort(int val) {
590     return String.valueOf(val);
591 }
592
593 public String printAnySimpleType(String val) {
594     return val;
595 }
596
597 /**
598  * Just return the string passed as a parameter but
599  * installs an instance of this class as the DatatypeConverter
600  * implementation. Used from static fixed value initializers.
601  */
602 public static String installHook(String s) {
603     DatatypeConverter.setDatatypeConverter(theInstance);
604     return s;
605 }
606 // base64 decoder
607 private static final byte[] decodeMap = initDecodeMap();
608 private static final byte PADDING = 127;
609
610 private static byte[] initDecodeMap() {
611     byte[] map = new byte[128];
612     int i;
613     for (i = 0; i < 128; i++) {
614         map[i] = -1;
615     }
616
617     for (i = 'A'; i <= 'Z'; i++) {
618         map[i] = (byte) (i - 'A');
619     }
620     for (i = 'a'; i <= 'z'; i++) {
621         map[i] = (byte) (i - 'a' + 26);
622     }
623     for (i = '0'; i <= '9'; i++) {
624         map[i] = (byte) (i - '0' + 52);
625     }
626     map['+'] = 62;
```



```

627     map['/'] = 63;
628     map['='] = PADDING;
629
630     return map;
631 }
632
633 /**
634  * computes the length of binary data speculatively.
635  *
636  * <p>
637  * Our requirement is to create byte[] of the exact length to store the binary data.
638  * If we do this in a straight-forward way, it takes two passes over the data.
639  * Experiments show that this is a non-trivial overhead (35% or so is spent on
640  * the first pass in calculating the length.)
641  *
642  * <p>
643  * So the approach here is that we compute the length speculatively, without looking
644  * at the whole contents. The obtained speculative value is never less than the
645  * actual length of the binary data, but it may be bigger. So if the speculation
646  * goes wrong, we'll pay the cost of reallocation and buffer copying.
647  *
648  * <p>
649  * If the base64 text is tightly packed with no indentation nor illegal char
650  * (like what most web services produce), then the speculation of this method
651  * will be correct, so we get the performance benefit.
652  */
653 private static int guessLength(String text) {
654     final int len = text.length();
655
656     // compute the tail '=' chars
657     int j = len - 1;
658     for (; j >= 0; j--) {
659         byte code = decodeMap[text.charAt(j)];
660         if (code == PADDING) {
661             continue;
662         }
663         if (code == -1) // most likely this base64 text is indented. go with the upper bound
664         {
665             return text.length() / 4 * 3;
666         }
667         break;
668     }
669
670     j++; // text.charAt(j) is now at some base64 char, so +1 to make it the size
671     int padSize = len - j;
672     if (padSize > 2) // something is wrong with base64. be safe and go with the upper bound
673     {
674         return text.length() / 4 * 3;
675     }
676
677     // so far this base64 looks like it's unindented tightly packed base64.
678     // take a chance and create an array with the expected size
679     return text.length() / 4 * 3 - padSize;

```

```

680     }
681
682     /**
683     * @param text
684     *     base64Binary data is likely to be long, and decoding requires
685     *     each character to be accessed twice (once for counting length, another
686     *     for decoding.)
687     *
688     *     A benchmark showed that taking {@link String} is faster, presumably
689     *     because JIT can inline a lot of string access (with data of 1K chars, it was twice as
690     *     fast)
691     */
692     public static byte[] _parseBase64Binary(String text) {
693         final int buflen = guessLength(text);
694         final byte[] out = new byte[buflen];
695         int o = 0;
696
697         final int len = text.length();
698         int i;
699
700         final byte[] quadruplet = new byte[4];
701         int q = 0;
702
703         // convert each quadruplet to three bytes.
704         for (i = 0; i < len; i++) {
705             char ch = text.charAt(i);
706             byte v = decodeMap[ch];
707
708             if (v != -1) {
709                 quadruplet[q++] = v;
710             }
711
712             if (q == 4) {
713                 // quadruplet is now filled.
714                 out[o++] = (byte) ((quadruplet[0] << 2) | (quadruplet[1] >> 4));
715                 if (quadruplet[2] != PADDING) {
716                     out[o++] = (byte) ((quadruplet[1] << 4) | (quadruplet[2] >> 2));
717                 }
718                 if (quadruplet[3] != PADDING) {
719                     out[o++] = (byte) ((quadruplet[2] << 6) | (quadruplet[3]));
720                 }
721                 q = 0;
722             }
723         }
724
725         if (buflen == o) // speculation worked out to be OK
726         {
727             return out;
728         }
729
730         // we overestimated, so need to create a new buffer
731         byte[] nb = new byte[o];
732         System.arraycopy(out, 0, nb, 0, o);

```

```

732     return nb;
733 }
734 private static final char[] encodeMap = initEncodeMap();
735
736 private static char[] initEncodeMap() {
737     char[] map = new char[64];
738     int i;
739     for (i = 0; i < 26; i++) {
740         map[i] = (char) ('A' + i);
741     }
742     for (i = 26; i < 52; i++) {
743         map[i] = (char) ('a' + (i - 26));
744     }
745     for (i = 52; i < 62; i++) {
746         map[i] = (char) ('0' + (i - 52));
747     }
748     map[62] = '+';
749     map[63] = '/';
750
751     return map;
752 }
753
754 public static char encode(int i) {
755     return encodeMap[i & 0x3F];
756 }
757
758 public static byte encodeByte(int i) {
759     return (byte) encodeMap[i & 0x3F];
760 }
761
762 public static String _printBase64Binary(byte[] input) {
763     return _printBase64Binary(input, 0, input.length);
764 }
765
766 public static String _printBase64Binary(byte[] input, int offset, int len) {
767     char[] buf = new char[((len + 2) / 3) * 4];
768     int ptr = _printBase64Binary(input, offset, len, buf, 0);
769     assert ptr == buf.length;
770     return new String(buf);
771 }
772
773 /**
774  * Encodes a byte array into a char array by doing base64 encoding.
775  *
776  * The caller must supply a big enough buffer.
777  *
778  * @return
779  *     the value of {@code ptr+((len+2)/3)*4}, which is the new offset
780  *     in the output buffer where the further bytes should be placed.
781  */
782 public static int _printBase64Binary(byte[] input, int offset, int len, char[] buf, int ptr) {
783     // encode elements until only 1 or 2 elements are left to encode
784     int remaining = len;

```

```

785     int i;
786     for (i = offset; remaining >= 3; remaining -= 3, i += 3) {
787         buf[ptr++] = encode(input[i] >> 2);
788         buf[ptr++] = encode(
789             ((input[i] & 0x3) << 4)
790             | ((input[i + 1] >> 4) & 0xF));
791         buf[ptr++] = encode(
792             ((input[i + 1] & 0xF) << 2)
793             | ((input[i + 2] >> 6) & 0x3));
794         buf[ptr++] = encode(input[i + 2] & 0x3F);
795     }
796     // encode when exactly 1 element (left) to encode
797     if (remaining == 1) {
798         buf[ptr++] = encode(input[i] >> 2);
799         buf[ptr++] = encode(((input[i] & 0x3) << 4));
800         buf[ptr++] = '=';
801         buf[ptr++] = '=';
802     }
803     // encode when exactly 2 elements (left) to encode
804     if (remaining == 2) {
805         buf[ptr++] = encode(input[i] >> 2);
806         buf[ptr++] = encode(((input[i] & 0x3) << 4)
807             | ((input[i + 1] >> 4) & 0xF));
808         buf[ptr++] = encode((input[i + 1] & 0xF) << 2);
809         buf[ptr++] = '=';
810     }
811     return ptr;
812 }
813
814 /**
815  * Encodes a byte array into another byte array by first doing base64 encoding
816  * then encoding the result in ASCII.
817  *
818  * The caller must supply a big enough buffer.
819  *
820  * @return
821  *     the value of {@code ptr+((len+2)/3)*4}, which is the new offset
822  *     in the output buffer where the further bytes should be placed.
823  */
824 public static int _printBase64Binary(byte[] input, int offset, int len, byte[] out, int ptr) {
825     byte[] buf = out;
826     int remaining = len;
827     int i;
828     for (i=offset; remaining >= 3; remaining -= 3, i += 3 ) {
829         buf[ptr++] = encodeByte(input[i]>>2);
830         buf[ptr++] = encodeByte(
831             ((input[i]&0x3)<<4) |
832             ((input[i+1]>>4)&0xF));
833         buf[ptr++] = encodeByte(
834             ((input[i+1]&0xF)<<2)|
835             ((input[i+2]>>6)&0x3));
836         buf[ptr++] = encodeByte(input[i+2]&0x3F);
837     }

```

```

838     // encode when exactly 1 element (left) to encode
839     if (remaining == 1) {
840         buf[ptr++] = encodeByte(input[i]>>2);
841         buf[ptr++] = encodeByte(((input[i]&0x3)<<4);
842         buf[ptr++] = '=';
843         buf[ptr++] = '=';
844     }
845     // encode when exactly 2 elements (left) to encode
846     if (remaining == 2) {
847         buf[ptr++] = encodeByte(input[i]>>2);
848         buf[ptr++] = encodeByte(
849             ((input[i]&0x3)<<4) |
850             ((input[i+1]>>4)&0xF));
851         buf[ptr++] = encodeByte((input[i+1]&0xF)<<2);
852         buf[ptr++] = '=';
853     }
854
855     return ptr;
856 }
857
858 private static CharSequence removeOptionalPlus(CharSequence s) {
859     int len = s.length();
860
861     if (len <= 1 || s.charAt(0) != '+') {
862         return s;
863     }
864
865     s = s.subSequence(1, len);
866     char ch = s.charAt(0);
867     if ('0' <= ch && ch <= '9') {
868         return s;
869     }
870     if ('.' == ch) {
871         return s;
872     }
873
874     throw new NumberFormatException();
875 }
876
877 private static boolean isDigitOrPeriodOrSign(char ch) {
878     if ('0' <= ch && ch <= '9') {
879         return true;
880     }
881     if (ch == '+' || ch == '-' || ch == '.') {
882         return true;
883     }
884     return false;
885 }
886 private static final DatatypeFactory datatypeFactory;
887
888 static {
889     try {
890         datatypeFactory = DatatypeFactory.newInstance();

```

```

891     } catch (DatatypeConfigurationException e) {
892         throw new Error(e);
893     }
894 }
895
896 private static final class CalendarFormatter {
897
898     public static String doFormat(String format, Calendar cal) throws IllegalArgumentException
899     {
900         int fidx = 0;
901         int flen = format.length();
902         StringBuilder buf = new StringBuilder();
903
904         while (fidx < flen) {
905             char fch = format.charAt(fidx++);
906
907             if (fch != '%') { // not a meta character
908                 buf.append(fch);
909                 continue;
910             }
911
912             // seen meta character. we don't do error check against the format
913             switch (format.charAt(fidx++)) {
914                 case 'Y': // year
915                     formatYear(cal, buf);
916                     break;
917
918                 case 'M': // month
919                     formatMonth(cal, buf);
920                     break;
921
922                 case 'D': // days
923                     formatDays(cal, buf);
924                     break;
925
926                 case 'h': // hours
927                     formatHours(cal, buf);
928                     break;
929
930                 case 'm': // minutes
931                     formatMinutes(cal, buf);
932                     break;
933
934                 case 's': // parse seconds.
935                     formatSeconds(cal, buf);
936                     break;
937
938                 case 'z': // time zone
939                     formatTimeZone(cal, buf);
940                     break;
941
942                 default:
943                     // illegal meta character. impossible.

```

```
943         throw new InternalError();
944     }
945 }
946
947     return buf.toString();
948 }
949
950 private static void formatYear(Calendar cal, StringBuilder buf) {
951     int year = cal.get(Calendar.YEAR);
952
953     String s;
954     if (year <= 0) // negative value
955     {
956         s = Integer.toString(1 - year);
957     } else // positive value
958     {
959         s = Integer.toString(year);
960     }
961
962     while (s.length() < 4) {
963         s = '0' + s;
964     }
965     if (year <= 0) {
966         s = '-' + s;
967     }
968
969     buf.append(s);
970 }
971
972 private static void formatMonth(Calendar cal, StringBuilder buf) {
973     formatTwoDigits(cal.get(Calendar.MONTH) + 1, buf);
974 }
975
976 private static void formatDays(Calendar cal, StringBuilder buf) {
977     formatTwoDigits(cal.get(Calendar.DAY_OF_MONTH), buf);
978 }
979
980 private static void formatHours(Calendar cal, StringBuilder buf) {
981     formatTwoDigits(cal.get(Calendar.HOUR_OF_DAY), buf);
982 }
983
984 private static void formatMinutes(Calendar cal, StringBuilder buf) {
985     formatTwoDigits(cal.get(Calendar.MINUTE), buf);
986 }
987
988 private static void formatSeconds(Calendar cal, StringBuilder buf) {
989     formatTwoDigits(cal.get(Calendar.SECOND), buf);
990     if (cal.isSet(Calendar.MILLISECOND)) { // milliseconds
991         int n = cal.get(Calendar.MILLISECOND);
992         if (n != 0) {
993             String ms = Integer.toString(n);
994             while (ms.length() < 3) {
995                 ms = '0' + ms; // left 0 paddings.
```

```

996         }
997         buf.append('.');
998         buf.append(ms);
999     }
1000 }
1001 }
1002
1003 /** formats time zone specifier. */
1004 private static void formatTimeZone(Calendar cal, StringBuilder buf) {
1005     TimeZone tz = cal.getTimeZone();
1006
1007     if (tz == null) {
1008         return;
1009     }
1010
1011     // otherwise print out normally.
1012     int offset = tz.getOffset(cal.getTime().getTime());
1013
1014     if (offset == 0) {
1015         buf.append('Z');
1016         return;
1017     }
1018
1019     if (offset >= 0) {
1020         buf.append('+');
1021     } else {
1022         buf.append('-');
1023         offset *= -1;
1024     }
1025
1026     offset /= 60 * 1000; // offset is in milli-seconds
1027
1028     formatTwoDigits(offset / 60, buf);
1029     buf.append(':');
1030     formatTwoDigits(offset % 60, buf);
1031 }
1032
1033 /** formats Integer into two-character-wide string. */
1034 private static void formatTwoDigits(int n, StringBuilder buf) {
1035     // n is always non-negative.
1036     if (n < 10) {
1037         buf.append('0');
1038     }
1039     buf.append(n);
1040 }
1041 }
1042 }

```

## 2.6 DatatypeConverterInterface.java

Secuencia 8: javax/xml/bind/DatatypeConverterInterface.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.

```



```
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 /**
29  * <p>
30  * The DatatypeConverterInterface is for JAXB provider use only. A
31  * JAXB provider must supply a class that implements this interface.
32  * JAXB Providers are required to call the
33  * {@link DatatypeConverter#setDatatypeConverter(DatatypeConverterInterface)
34  * DatatypeConverter.setDatatypeConverter} api at
35  * some point before the first marshal or unmarshal operation (perhaps during
36  * the call to JAXBContext.newInstance). This step is necessary to configure
37  * the converter that should be used to perform the print and parse
38  * functionality. Calling this api repeatedly will have no effect - the
39  * DatatypeConverter instance passed into the first invocation is the one that
40  * will be used from then on.
41  * </p>
42  *
43  * <p>
44  * This interface defines the parse and print methods. There is one
45  * parse and print method for each XML schema datatype specified in the
46  * the default binding Table 5-1 in the JAXB specification.
47  * </p>
48  *
49  * <p>
50  * The parse and print methods defined here are invoked by the static parse
51  * and print methods defined in the {@link DatatypeConverter DatatypeConverter}
52  * class.
53  * </p>
54  *
55  * <p>
```

```

56 * A parse method for a XML schema datatype must be capable of converting any
57 * lexical representation of the XML schema datatype ( specified by the
58 * <a href="http://www.w3.org/TR/xmlschema-2/"> XML Schema Part2: Datatypes
59 * specification</a> into a value in the value space of the XML schema datatype.
60 * If an error is encountered during conversion, then an IllegalArgumentException
61 * or a subclass of IllegalArgumentException must be thrown by the method.
62 *
63 * </p>
64 *
65 * <p>
66 * A print method for a XML schema datatype can output any lexical
67 * representation that is valid with respect to the XML schema datatype.
68 * If an error is encountered during conversion, then an IllegalArgumentException,
69 * or a subclass of IllegalArgumentException must be thrown by the method.
70 * </p>
71 *
72 * The prefix xsd: is used to refer to XML schema datatypes
73 * <a href="http://www.w3.org/TR/xmlschema-2/"> XML Schema Part2: Datatypes
74 * specification.</a>
75 *
76 * <p>
77 * @author <ul><li>Sekhar Vajjhala, Sun Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems Inc
      .</li><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Ryan Shoemaker, Sun Microsystems
      Inc.</li></ul>
78 * @see DatatypeConverter
79 * @see ParseConversionEvent
80 * @see PrintConversionEvent
81 * @since JAXB1.0
82 */
83
84 public interface DatatypeConverterInterface {
85     /**
86      * <p>
87      * Convert the string argument into a string.
88      * @param lexicalXSDString
89      *     A lexical representation of the XML Schema datatype xsd:string
90      * @return
91      *     A string that is the same as the input string.
92      */
93     public String parseString( String lexicalXSDString );
94
95     /**
96      * <p>
97      * Convert the string argument into a BigInteger value.
98      * @param lexicalXSDInteger
99      *     A string containing a lexical representation of
100      *     xsd:integer.
101      * @return
102      *     A BigInteger value represented by the string argument.
103      * @throws NumberFormatException <code>lexicalXSDInteger</code> is not a valid string
      representation of a {@link java.math.BigInteger} value.
104      */
105     public java.math.BigInteger parseInteger( String lexicalXSDInteger );

```

```
106
107 /**
108  * <p>
109  * Convert the string argument into an int value.
110  * @param lexicalXSDInt
111  *     A string containing a lexical representation of
112  *     xsd:int.
113  * @return
114  *     An int value represented by the string argument.
115  * @throws NumberFormatException <code>lexicalXSDInt</code> is not a valid string
116  *     representation of an <code>int</code> value.
117  */
118 public int parseInt( String lexicalXSDInt );
119
120 /**
121  * <p>
122  * Converts the string argument into a long value.
123  * @param lexicalXSDDLong
124  *     A string containing lexical representation of
125  *     xsd:long.
126  * @return
127  *     A long value represented by the string argument.
128  * @throws NumberFormatException <code>lexicalXSDDLong</code> is not a valid string
129  *     representation of a <code>long</code> value.
130  */
131 public long parseLong( String lexicalXSDDLong );
132
133 /**
134  * <p>
135  * Converts the string argument into a short value.
136  * @param lexicalXSDDShort
137  *     A string containing lexical representation of
138  *     xsd:short.
139  * @return
140  *     A short value represented by the string argument.
141  * @throws NumberFormatException <code>lexicalXSDDShort</code> is not a valid string
142  *     representation of a <code>short</code> value.
143  */
144 public short parseShort( String lexicalXSDDShort );
145
146 /**
147  * <p>
148  * Converts the string argument into a BigDecimal value.
149  * @param lexicalXSDDDecimal
150  *     A string containing lexical representation of
151  *     xsd:decimal.
152  * @return
153  *     A BigDecimal value represented by the string argument.
154  * @throws NumberFormatException <code>lexicalXSDDDecimal</code> is not a valid string
155  *     representation of {@link java.math.BigDecimal}.
156  */
157 public java.math.BigDecimal parseDecimal( String lexicalXSDDDecimal );
```

```
155 /**
156  * <p>
157  * Converts the string argument into a float value.
158  * @param lexicalXSDFloat
159  *     A string containing lexical representation of
160  *     xsd:float.
161  * @return
162  *     A float value represented by the string argument.
163  * @throws NumberFormatException <code>lexicalXSDFloat</code> is not a valid string
164  *     representation of a <code>float</code> value.
165  */
166 public float parseFloat( String lexicalXSDFloat );
167 /**
168  * <p>
169  * Converts the string argument into a double value.
170  * @param lexicalXSDDouble
171  *     A string containing lexical representation of
172  *     xsd:double.
173  * @return
174  *     A double value represented by the string argument.
175  * @throws NumberFormatException <code>lexicalXSDDouble</code> is not a valid string
176  *     representation of a <code>double</code> value.
177  */
178 public double parseDouble( String lexicalXSDDouble );
179 /**
180  * <p>
181  * Converts the string argument into a boolean value.
182  * @param lexicalXSDBoolean
183  *     A string containing lexical representation of
184  *     xsd:boolean.
185  * @return
186  *     A boolean value represented by the string argument.
187  * @throws IllegalArgumentException if string parameter does not conform to lexical value space
188  *     defined in XML Schema Part 2: Datatypes for xsd:boolean.
189  */
190 public boolean parseBoolean( String lexicalXSDBoolean );
191 /**
192  * <p>
193  * Converts the string argument into a byte value.
194  * @param lexicalXSDByte
195  *     A string containing lexical representation of
196  *     xsd:byte.
197  * @return
198  *     A byte value represented by the string argument.
199  * @throws NumberFormatException <code>lexicalXSDByte</code> does not contain a parseable byte.
200  * @throws IllegalArgumentException if string parameter does not conform to lexical value space
201  *     defined in XML Schema Part 2: Datatypes for xsd:byte.
202  */
203 public byte parseByte( String lexicalXSDByte );
```

```

204  /**
205   * <p>
206   * Converts the string argument into a QName value.
207   *
208   * <p>
209   * String parameter <tt>lexicalXSDQname</tt> must conform to lexical value space specifed at
210   * <a href="http://www.w3.org/TR/xmlschema-2/#QName">XML Schema Part 2:Datatypes specification:
211   *   QNames</a>
212   *
213   * @param lexicalXSDQName
214   *   A string containing lexical representation of xsd:QName.
215   * @param nsc
216   *   A namespace context for interpreting a prefix within a QName.
217   * @return
218   *   A QName value represented by the string argument.
219   * @throws IllegalArgumentException if string parameter does not conform to XML Schema Part 2
220   *   specification or
221   *   if namespace prefix of <tt>lexicalXSDQname</tt> is not bound to a URI in
222   *   NamespaceContext <tt>nsc</tt>.
223   */
224  public javax.xml.namespace.QName parseQName( String lexicalXSDQName,
225                                              javax.xml.namespace.NamespaceContext nsc);
226
227  /**
228   * <p>
229   * Converts the string argument into a Calendar value.
230   * @param lexicalXSDDateTime
231   *   A string containing lexical representation of
232   *   xsd:datetime.
233   * @return
234   *   A Calendar object represented by the string argument.
235   * @throws IllegalArgumentException if string parameter does not conform to lexical value space
236   *   defined in XML Schema Part 2: Datatypes for xsd:dateTime.
237   */
238  public java.util.Calendar parseDateTime( String lexicalXSDDateTime );
239
240  /**
241   * <p>
242   * Converts the string argument into an array of bytes.
243   * @param lexicalXSDBase64Binary
244   *   A string containing lexical representation
245   *   of xsd:base64Binary.
246   * @return
247   *   An array of bytes represented by the string argument.
248   * @throws IllegalArgumentException if string parameter does not conform to lexical value space
249   *   defined in XML Schema Part 2: Datatypes for xsd:base64Binary
250   */
251  public byte[] parseBase64Binary( String lexicalXSDBase64Binary );

```

```
252     *    A string containing lexical representation of
253     *    xsd:hexBinary.
254     * @return
255     *    An array of bytes represented by the string argument.
256     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
257     *    defined in XML Schema Part 2: Datatypes for xsd:hexBinary.
258     */
259     public byte[] parseHexBinary( String lexicalXSDHexBinary );
260
261     /**
262     * <p>
263     * Converts the string argument into a long value.
264     * @param lexicalXSDUnsignedInt
265     *    A string containing lexical representation
266     *    of xsd:unsignedInt.
267     * @return
268     *    A long value represented by the string argument.
269     * @throws NumberFormatException if string parameter can not be parsed into a <tt>long</tt>
270     *    value.
271     */
272     public long parseUnsignedInt( String lexicalXSDUnsignedInt );
273
274     /**
275     * <p>
276     * Converts the string argument into an int value.
277     * @param lexicalXSDUnsignedShort
278     *    A string containing lexical
279     *    representation of xsd:unsignedShort.
280     * @return
281     *    An int value represented by the string argument.
282     * @throws NumberFormatException if string parameter can not be parsed into an <tt>int</tt>
283     *    value.
284     */
285     public int parseUnsignedShort( String lexicalXSDUnsignedShort );
286
287     /**
288     * <p>
289     * Converts the string argument into a Calendar value.
290     * @param lexicalXSDTime
291     *    A string containing lexical representation of
292     *    xsd:Time.
293     * @return
294     *    A Calendar value represented by the string argument.
295     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
296     *    defined in XML Schema Part 2: Datatypes for xsd:Time.
297     */
298     public java.util.Calendar parseTime( String lexicalXSDTime );
299
300     /**
301     * <p>
302     * Converts the string argument into a Calendar value.
303     * @param lexicalXSDDate
304     *    A string containing lexical representation of
```

```
301     *    xsd:Date.
302     * @return
303     *    A Calendar value represented by the string argument.
304     * @throws IllegalArgumentException if string parameter does not conform to lexical value space
305     *    defined in XML Schema Part 2: Datatypes for xsd:Date.
306     */
307     public java.util.Calendar parseDate( String lexicalXSDDate );
308
309     /**
310     * <p>
311     * Return a string containing the lexical representation of the
312     * simple type.
313     * @param lexicalXSDAnySimpleType
314     *    A string containing lexical
315     *    representation of the simple type.
316     * @return
317     *    A string containing the lexical representation of the
318     *    simple type.
319     */
320     public String parseAnySimpleType( String lexicalXSDAnySimpleType );
321
322     /**
323     * <p>
324     * Converts the string argument into a string.
325     * @param val
326     *    A string value.
327     * @return
328     *    A string containing a lexical representation of xsd:string
329     */
330     public String printString( String val );
331
332     /**
333     * <p>
334     * Converts a BigInteger value into a string.
335     * @param val
336     *    A BigInteger value
337     * @return
338     *    A string containing a lexical representation of xsd:integer
339     * @throws IllegalArgumentException <tt>val</tt> is null.
340     */
341     public String printInteger( java.math.BigInteger val );
342
343     /**
344     * <p>
345     * Converts an int value into a string.
346     * @param val
347     *    An int value
348     * @return
349     *    A string containing a lexical representation of xsd:int
350     */
351     public String printInt( int val );
352
```

```
353 /**
354  * <p>
355  * Converts a long value into a string.
356  * @param val
357  *     A long value
358  * @return
359  *     A string containing a lexical representation of xsd:long
360  */
361 public String printLong( long val );
362
363 /**
364  * <p>
365  * Converts a short value into a string.
366  * @param val
367  *     A short value
368  * @return
369  *     A string containing a lexical representation of xsd:short
370  */
371 public String printShort( short val );
372
373 /**
374  * <p>
375  * Converts a BigDecimal value into a string.
376  * @param val
377  *     A BigDecimal value
378  * @return
379  *     A string containing a lexical representation of xsd:decimal
380  * @throws IllegalArgumentException <tt>val</tt> is null.
381  */
382 public String printDecimal( java.math.BigDecimal val );
383
384 /**
385  * <p>
386  * Converts a float value into a string.
387  * @param val
388  *     A float value
389  * @return
390  *     A string containing a lexical representation of xsd:float
391  */
392 public String printFloat( float val );
393
394 /**
395  * <p>
396  * Converts a double value into a string.
397  * @param val
398  *     A double value
399  * @return
400  *     A string containing a lexical representation of xsd:double
401  */
402 public String printDouble( double val );
403
404 /**
405  * <p>
```



```
406     * Converts a boolean value into a string.
407     * @param val
408     *     A boolean value
409     * @return
410     *     A string containing a lexical representation of xsd:boolean
411     */
412     public String printBoolean( boolean val );
413
414     /**
415     * <p>
416     * Converts a byte value into a string.
417     * @param val
418     *     A byte value
419     * @return
420     *     A string containing a lexical representation of xsd:byte
421     */
422     public String printByte( byte val );
423
424     /**
425     * <p>
426     * Converts a QName instance into a string.
427     * @param val
428     *     A QName value
429     * @param nsc
430     *     A namespace context for interpreting a prefix within a QName.
431     * @return
432     *     A string containing a lexical representation of QName
433     * @throws IllegalArgumentException if <tt>val</tt> is null or
434     * if <tt>nsc</tt> is non-null or <tt>nsc.getPrefix(nsprefixFromVal)</tt> is null.
435     */
436     public String printQName( javax.xml.namespace.QName val,
437                               javax.xml.namespace.NamespaceContext nsc );
438
439     /**
440     * <p>
441     * Converts a Calendar value into a string.
442     * @param val
443     *     A Calendar value
444     * @return
445     *     A string containing a lexical representation of xsd:dateTime
446     * @throws IllegalArgumentException if <tt>val</tt> is null.
447     */
448     public String printDateTime( java.util.Calendar val );
449
450     /**
451     * <p>
452     * Converts an array of bytes into a string.
453     * @param val
454     *     an array of bytes
455     * @return
456     *     A string containing a lexical representation of xsd:base64Binary
457     * @throws IllegalArgumentException if <tt>val</tt> is null.
458     */
```

```
459 public String printBase64Binary( byte[] val );
460
461 /**
462  * <p>
463  * Converts an array of bytes into a string.
464  * @param val
465  *     an array of bytes
466  * @return
467  *     A string containing a lexical representation of xsd:hexBinary
468  * @throws IllegalArgumentException if <tt>val</tt> is null.
469  */
470 public String printHexBinary( byte[] val );
471
472 /**
473  * <p>
474  * Converts a long value into a string.
475  * @param val
476  *     A long value
477  * @return
478  *     A string containing a lexical representation of xsd:unsignedInt
479  */
480 public String printUnsignedInt( long val );
481
482 /**
483  * <p>
484  * Converts an int value into a string.
485  * @param val
486  *     An int value
487  * @return
488  *     A string containing a lexical representation of xsd:unsignedShort
489  */
490 public String printUnsignedShort( int val );
491
492 /**
493  * <p>
494  * Converts a Calendar value into a string.
495  * @param val
496  *     A Calendar value
497  * @return
498  *     A string containing a lexical representation of xsd:time
499  * @throws IllegalArgumentException if <tt>val</tt> is null.
500  */
501 public String printTime( java.util.Calendar val );
502
503 /**
504  * <p>
505  * Converts a Calendar value into a string.
506  * @param val
507  *     A Calendar value
508  * @return
509  *     A string containing a lexical representation of xsd:date
510  * @throws IllegalArgumentException if <tt>val</tt> is null.
511  */
```

```

512 public String printDate( java.util.Calendar val );
513
514 /**
515  * <p>
516  * Converts a string value into a string.
517  * @param val
518  *     A string value
519  * @return
520  *     A string containing a lexical representation of xsd:AnySimpleType
521  */
522 public String printAnySimpleType( String val );
523 }

```

## 2.7 Element.java

### Secuencia 9: javax/xml/bind/Element.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 /**
29  * This is an element marker interface.
30  *
31  * Under certain circumstances, it is necessary for the binding compiler to
32  * generate derived java content classes that implement this interface. In
33  * those cases, client applications must supply element instances rather than
34  * types of elements. For more detail, see section 5.7 "Element Declaration"
35  * and 5.7.1 "Bind to Java Element Interface" of the specification.
36  *

```

```

37  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
      Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
38  * @since JAXB1.0
39  */
40
41  public interface Element {
42  }

```

## 2.8 GetPropertyAction.java

Secuencia 10: javax/xml/bind/GetPropertyAction.java

```

1  /*
2  * Copyright (c) 2006, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import java.security.PrivilegedAction;
29
30 /**
31  * {@link PrivilegedAction} that gets the system property value.
32  * @author Kohsuke Kawaguchi
33  */
34 final class GetPropertyAction implements PrivilegedAction<String> {
35     private final String propertyName;
36
37     public GetPropertyAction(String propertyName) {
38         this.propertyName = propertyName;
39     }
40
41     public String run() {
42         return System.getProperty(propertyName);

```

```
43     }  
44 }
```

## 2.9 JAXB.java

Secuencia 11: javax/xml/bind/JAXB.java

```
1  /*  
2  * Copyright (c) 2006, 2013, Oracle and/or its affiliates. All rights reserved.  
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.  
4  *  
5  *  
6  *  
7  *  
8  *  
9  *  
10 *  
11 *  
12 *  
13 *  
14 *  
15 *  
16 *  
17 *  
18 *  
19 *  
20 *  
21 *  
22 *  
23 *  
24 */  
25  
26 package javax.xml.bind;  
27  
28 import javax.xml.bind.annotation.XmlRootElement;  
29 import javax.xml.namespace.QName;  
30 import javax.xml.transform.Result;  
31 import javax.xml.transform.Source;  
32 import javax.xml.transform.stream.StreamResult;  
33 import javax.xml.transform.stream.StreamSource;  
34 import java.beans.Introspector;  
35 import java.io.File;  
36 import java.io.IOException;  
37 import java.io.InputStream;  
38 import java.io.OutputStream;  
39 import java.io.Reader;  
40 import java.io.Writer;  
41 import java.lang.ref.WeakReference;  
42 import java.net.HttpURLConnection;  
43 import java.net.URI;  
44 import java.net.URISyntaxException;  
45 import java.net.URL;  
46 import java.net.URLConnection;  
47
```

```

48 /**
49  * Class that defines convenience methods for common, simple use of JAXB.
50  *
51  * <p>
52  * Methods defined in this class are convenience methods that combine several basic operations
53  * in the {@link JAXBContext}, {@link Unmarshaller}, and {@link Marshaller}.
54  *
55  * They are designed
56  * to be the preferred methods for developers new to JAXB. They have
57  * the following characteristics:
58  *
59  * <ol>
60  * <li>Generally speaking, the performance is not necessarily optimal.
61  *     It is expected that people who need to write performance
62  *     critical code will use the rest of the JAXB API directly.
63  * <li>Errors that happen during the processing is wrapped into
64  *     {@link DataBindingException} (which will have {@link JAXBException}
65  *     as its {@link Throwable#getCause() cause}. It is expected that
66  *     people who prefer the checked exception would use
67  *     the rest of the JAXB API directly.
68  * </ol>
69  *
70  * <p>
71  * In addition, the <tt>unmarshal</tt> methods have the following characteristic:
72  *
73  * <ol>
74  * <li>Schema validation is not performed on the input XML.
75  *     The processing will try to continue even if there
76  *     are errors in the XML, as much as possible. Only as
77  *     the last resort, this method fails with {@link DataBindingException}.
78  * </ol>
79  *
80  * <p>
81  * Similarly, the <tt>marshal</tt> methods have the following characteristic:
82  * <ol>
83  * <li>The processing will try to continue even if the Java object tree
84  *     does not meet the validity requirement. Only as
85  *     the last resort, this method fails with {@link DataBindingException}.
86  * </ol>
87  *
88  *
89  * <p>
90  * All the methods on this class require non-null arguments to all parameters.
91  * The <tt>unmarshal</tt> methods either fail with an exception or return
92  * a non-null value.
93  *
94  * @author Kohsuke Kawaguchi
95  * @since 2.1
96  */
97 public final class JAXB {
98     /**
99     * No instantiation is allowed.
100    */

```

```

101 private JAXB() {}
102
103 /**
104  * To improve the performance, we'll cache the last {@link JAXBContext} used.
105  */
106 private static final class Cache {
107     final Class type;
108     final JAXBContext context;
109
110     public Cache(Class type) throws JAXBException {
111         this.type = type;
112         this.context = JAXBContext.newInstance(type);
113     }
114 }
115
116 /**
117  * Cache. We don't want to prevent the {@link Cache#type} from GC-ed,
118  * hence {@link WeakReference}.
119  */
120 private static volatile WeakReference<Cache> cache;
121
122 /**
123  * Obtains the {@link JAXBContext} from the given type,
124  * by using the cache if possible.
125  *
126  * <p>
127  * We don't use locks to control access to {@link #cache}, but this code
128  * should be thread-safe thanks to the immutable {@link Cache} and {@code volatile}.
129  */
130 private static <T> JAXBContext getContext(Class<T> type) throws JAXBException {
131     WeakReference<Cache> c = cache;
132     if(c!=null) {
133         Cache d = c.get();
134         if(d!=null && d.type==type)
135             return d.context;
136     }
137
138     // overwrite the cache
139     Cache d = new Cache(type);
140     cache = new WeakReference<Cache>(d);
141
142     return d.context;
143 }
144
145 /**
146  * Reads in a Java object tree from the given XML input.
147  *
148  * @param xml
149  *     Reads the entire file as XML.
150  */
151 public static <T> T unmarshal( File xml, Class<T> type ) {
152     try {
153         JAXBElement<T> item = getContext(type).createUnmarshaller().unmarshal(new StreamSource(

```

```

        xml), type);
154     return item.getValue();
155 } catch (JAXBException e) {
156     throw new DataBindingException(e);
157 }
158 }
159
160 /**
161  * Reads in a Java object tree from the given XML input.
162  *
163  * @param xml
164  *     The resource pointed by the URL is read in its entirety.
165  */
166 public static <T> T unmarshal( URL xml, Class<T> type ) {
167     try {
168         JAXBElement<T> item = getContext(type).createUnmarshaller().unmarshal(toSource(xml),
169                                     type);
169         return item.getValue();
170     } catch (JAXBException e) {
171         throw new DataBindingException(e);
172     } catch (IOException e) {
173         throw new DataBindingException(e);
174     }
175 }
176
177 /**
178  * Reads in a Java object tree from the given XML input.
179  *
180  * @param xml
181  *     The URI is {@link URI#toURL()} turned into URL} and then
182  *     follows the handling of <tt>URL</tt>.
183  */
184 public static <T> T unmarshal( URI xml, Class<T> type ) {
185     try {
186         JAXBElement<T> item = getContext(type).createUnmarshaller().unmarshal(toSource(xml),
187                                     type);
187         return item.getValue();
188     } catch (JAXBException e) {
189         throw new DataBindingException(e);
190     } catch (IOException e) {
191         throw new DataBindingException(e);
192     }
193 }
194
195 /**
196  * Reads in a Java object tree from the given XML input.
197  *
198  * @param xml
199  *     The string is first interpreted as an absolute <tt>URI</tt>.
200  *     If it's not {@link URI#isAbsolute()} a valid absolute URI},
201  *     then it's interpreted as a <tt>File</tt>
202  */
203 public static <T> T unmarshal( String xml, Class<T> type ) {

```



```

204     try {
205         JAXBElement<T> item = getContext(type).createUnmarshaller().unmarshal(toSource(xml),
            type);
206         return item.getValue();
207     } catch (JAXBException e) {
208         throw new DataBindingException(e);
209     } catch (IOException e) {
210         throw new DataBindingException(e);
211     }
212 }
213
214 /**
215  * Reads in a Java object tree from the given XML input.
216  *
217  * @param xml
218  *     The entire stream is read as an XML infoset.
219  *     Upon a successful completion, the stream will be closed by this method.
220  */
221 public static <T> T unmarshal( InputStream xml, Class<T> type ) {
222     try {
223         JAXBElement<T> item = getContext(type).createUnmarshaller().unmarshal(toSource(xml),
            type);
224         return item.getValue();
225     } catch (JAXBException e) {
226         throw new DataBindingException(e);
227     } catch (IOException e) {
228         throw new DataBindingException(e);
229     }
230 }
231
232 /**
233  * Reads in a Java object tree from the given XML input.
234  *
235  * @param xml
236  *     The character stream is read as an XML infoset.
237  *     The encoding declaration in the XML will be ignored.
238  *     Upon a successful completion, the stream will be closed by this method.
239  */
240 public static <T> T unmarshal( Reader xml, Class<T> type ) {
241     try {
242         JAXBElement<T> item = getContext(type).createUnmarshaller().unmarshal(toSource(xml),
            type);
243         return item.getValue();
244     } catch (JAXBException e) {
245         throw new DataBindingException(e);
246     } catch (IOException e) {
247         throw new DataBindingException(e);
248     }
249 }
250
251 /**
252  * Reads in a Java object tree from the given XML input.
253  *

```

```

254     * @param xml
255     *     The XML infoset that the {@link Source} represents is read.
256     */
257     public static <T> T unmarshal( Source xml, Class<T> type ) {
258         try {
259             JAXBElement<T> item = getContext(type).createUnmarshaller().unmarshal(toSource(xml),
260                                     type);
261             return item.getValue();
262         } catch (JAXBException e) {
263             throw new DataBindingException(e);
264         } catch (IOException e) {
265             throw new DataBindingException(e);
266         }
267     }
268
269     /**
270     * Creates {@link Source} from various XML representation.
271     * See {@link #unmarshal} for the conversion rules.
272     */
273     private static Source toSource(Object xml) throws IOException {
274         if(xml==null)
275             throw new IllegalArgumentException("no XML is given");
276
277         if (xml instanceof String) {
278             try {
279                 xml=new URI((String)xml);
280             } catch (URISyntaxException e) {
281                 xml=new File((String)xml);
282             }
283         }
284
285         if (xml instanceof File) {
286             File file = (File) xml;
287             return new StreamSource(file);
288         }
289
290         if (xml instanceof URI) {
291             URI uri = (URI) xml;
292             xml=uri.toURL();
293         }
294
295         if (xml instanceof URL) {
296             URL url = (URL) xml;
297             return new StreamSource(url.toExternalForm());
298         }
299
300         if (xml instanceof InputStream) {
301             InputStream in = (InputStream) xml;
302             return new StreamSource(in);
303         }
304
305         if (xml instanceof Reader) {
306             Reader r = (Reader) xml;
307             return new StreamSource(r);
308         }
309
310         if (xml instanceof Source) {
311             return (Source) xml;
312         }
313     }

```

```

306         return (Source) xml;
307     }
308     throw new IllegalArgumentException("I don't understand how to handle "+xml.getClass());
309 }
310
311 /**
312  * Writes a Java object tree to XML and store it to the specified location.
313  *
314  * @param jaxbObject
315  *     The Java object to be marshalled into XML. If this object is
316  *     a {@link JAXBElement}, it will provide the root tag name and
317  *     the body. If this object has {@link XmlRootElement}
318  *     on its class definition, that will be used as the root tag name
319  *     and the given object will provide the body. Otherwise,
320  *     the root tag name is {@link Introspector#decapitalize(String) infered} from
321  *     {@link Class#getSimpleName() the short class name}.
322  *     This parameter must not be null.
323  *
324  * @param xml
325  *     XML will be written to this file. If it already exists,
326  *     it will be overwritten.
327  *
328  * @throws DataBindingException
329  *     If the operation fails, such as due to I/O error, unbindable classes.
330  */
331 public static void marshal( Object jaxbObject, File xml ) {
332     _marshal(jaxbObject,xml);
333 }
334
335 /**
336  * Writes a Java object tree to XML and store it to the specified location.
337  *
338  * @param jaxbObject
339  *     The Java object to be marshalled into XML. If this object is
340  *     a {@link JAXBElement}, it will provide the root tag name and
341  *     the body. If this object has {@link XmlRootElement}
342  *     on its class definition, that will be used as the root tag name
343  *     and the given object will provide the body. Otherwise,
344  *     the root tag name is {@link Introspector#decapitalize(String) infered} from
345  *     {@link Class#getSimpleName() the short class name}.
346  *     This parameter must not be null.
347  *
348  * @param xml
349  *     The XML will be {@link URLConnection#getOutputStream() sent} to the
350  *     resource pointed by this URL. Note that not all <tt>URL</tt>s support
351  *     such operation, and exact semantics depends on the <tt>URL</tt>
352  *     implementations. In case of {@link HttpURLConnection HTTP URLs},
353  *     this will perform HTTP POST.
354  *
355  * @throws DataBindingException
356  *     If the operation fails, such as due to I/O error, unbindable classes.
357  */
358 public static void marshal( Object jaxbObject, URL xml ) {

```

```

359     _marshal(jaxbObject,xml);
360 }
361
362 /**
363  * Writes a Java object tree to XML and store it to the specified location.
364  *
365  * @param jaxbObject
366  *     The Java object to be marshalled into XML. If this object is
367  *     a {@link JAXBElement}, it will provide the root tag name and
368  *     the body. If this object has {@link XmlRootElement}
369  *     on its class definition, that will be used as the root tag name
370  *     and the given object will provide the body. Otherwise,
371  *     the root tag name is {@link Introspector#decapitalize(String) infered} from
372  *     {@link Class#getSimpleName() the short class name}.
373  *     This parameter must not be null.
374  *
375  * @param xml
376  *     The URI is {@link URI#toURL() turned into URL} and then
377  *     follows the handling of <tt>URL</tt>. See above.
378  *
379  * @throws DataBindingException
380  *     If the operation fails, such as due to I/O error, unbindable classes.
381  */
382 public static void marshal( Object jaxbObject, URI xml ) {
383     _marshal(jaxbObject,xml);
384 }
385
386 /**
387  * Writes a Java object tree to XML and store it to the specified location.
388  *
389  * @param jaxbObject
390  *     The Java object to be marshalled into XML. If this object is
391  *     a {@link JAXBElement}, it will provide the root tag name and
392  *     the body. If this object has {@link XmlRootElement}
393  *     on its class definition, that will be used as the root tag name
394  *     and the given object will provide the body. Otherwise,
395  *     the root tag name is {@link Introspector#decapitalize(String) infered} from
396  *     {@link Class#getSimpleName() the short class name}.
397  *     This parameter must not be null.
398  *
399  * @param xml
400  *     The string is first interpreted as an absolute <tt>URI</tt>.
401  *     If it's not {@link URI#isAbsolute() a valid absolute URI},
402  *     then it's interpreted as a <tt>File</tt>
403  *
404  * @throws DataBindingException
405  *     If the operation fails, such as due to I/O error, unbindable classes.
406  */
407 public static void marshal( Object jaxbObject, String xml ) {
408     _marshal(jaxbObject,xml);
409 }
410
411 /**

```

```

412     * Writes a Java object tree to XML and store it to the specified location.
413     *
414     * @param jaxbObject
415     *     The Java object to be marshalled into XML. If this object is
416     *     a {@link JAXBElement}, it will provide the root tag name and
417     *     the body. If this object has {@link XmlRootElement}
418     *     on its class definition, that will be used as the root tag name
419     *     and the given object will provide the body. Otherwise,
420     *     the root tag name is {@link Introspector#decapitalize(String) inferred} from
421     *     {@link Class#getSimpleName()} the short class name}.
422     *     This parameter must not be null.
423     *
424     * @param xml
425     *     The XML will be sent to the given {@link OutputStream}.
426     *     Upon a successful completion, the stream will be closed by this method.
427     *
428     * @throws DataBindingException
429     *     If the operation fails, such as due to I/O error, unbindable classes.
430     */
431     public static void marshal( Object jaxbObject, OutputStream xml ) {
432         _marshal(jaxbObject,xml);
433     }
434
435     /**
436     * Writes a Java object tree to XML and store it to the specified location.
437     *
438     * @param jaxbObject
439     *     The Java object to be marshalled into XML. If this object is
440     *     a {@link JAXBElement}, it will provide the root tag name and
441     *     the body. If this object has {@link XmlRootElement}
442     *     on its class definition, that will be used as the root tag name
443     *     and the given object will provide the body. Otherwise,
444     *     the root tag name is {@link Introspector#decapitalize(String) inferred} from
445     *     {@link Class#getSimpleName()} the short class name}.
446     *     This parameter must not be null.
447     *
448     * @param xml
449     *     The XML will be sent as a character stream to the given {@link Writer}.
450     *     Upon a successful completion, the stream will be closed by this method.
451     *
452     * @throws DataBindingException
453     *     If the operation fails, such as due to I/O error, unbindable classes.
454     */
455     public static void marshal( Object jaxbObject, Writer xml ) {
456         _marshal(jaxbObject,xml);
457     }
458
459     /**
460     * Writes a Java object tree to XML and store it to the specified location.
461     *
462     * @param jaxbObject
463     *     The Java object to be marshalled into XML. If this object is
464     *     a {@link JAXBElement}, it will provide the root tag name and

```

```

465 *     the body. If this object has {@link XmlRootElement}
466 *     on its class definition, that will be used as the root tag name
467 *     and the given object will provide the body. Otherwise,
468 *     the root tag name is {@link Introspector#decapitalize(String) inferred} from
469 *     {@link Class#getSimpleName() the short class name}.
470 *     This parameter must not be null.
471 *
472 * @param xml
473 *     The XML will be sent to the {@link Result} object.
474 *
475 * @throws DataBindingException
476 *     If the operation fails, such as due to I/O error, unbindable classes.
477 */
478 public static void marshal( Object jaxbObject, Result xml ) {
479     _marshal(jaxbObject,xml);
480 }
481
482 /**
483 * Writes a Java object tree to XML and store it to the specified location.
484 *
485 * <p>
486 * This method is a convenience method that combines several basic operations
487 * in the {@link JAXBContext} and {@link Marshaller}. This method is designed
488 * to be the preferred method for developers new to JAXB. This method
489 * has the following characteristics:
490 *
491 * <ol>
492 * <li>Generally speaking, the performance is not necessarily optimal.
493 *     It is expected that those people who need to write performance
494 *     critical code will use the rest of the JAXB API directly.
495 * <li>Errors that happen during the processing is wrapped into
496 *     {@link DataBindingException} (which will have {@link JAXBException}
497 *     as its {@link Throwable#getCause() cause}. It is expected that
498 *     those people who prefer the checked exception would use
499 *     the rest of the JAXB API directly.
500 * </ol>
501 *
502 * @param jaxbObject
503 *     The Java object to be marshalled into XML. If this object is
504 *     a {@link JAXBElement}, it will provide the root tag name and
505 *     the body. If this object has {@link XmlRootElement}
506 *     on its class definition, that will be used as the root tag name
507 *     and the given object will provide the body. Otherwise,
508 *     the root tag name is {@link Introspector#decapitalize(String) inferred} from
509 *     {@link Class#getSimpleName() the short class name}.
510 *     This parameter must not be null.
511 *
512 * @param xml
513 *     Represents the receiver of XML. Objects of the following types are allowed.
514 *
515 *     <table><tr>
516 *         <th>Type</th>
517 *         <th>Operation</th>

```

```

518 *      </tr><tr>
519 *          <td>{@link File}</td>
520 *          <td>XML will be written to this file. If it already exists,
521 *              it will be overwritten.</td>
522 *      </tr><tr>
523 *          <td>{@link URL}</td>
524 *          <td>The XML will be {@link URLConnection#getOutputStream() sent} to the
525 *              resource pointed by this URL. Note that not all <tt>URL</tt>s support
526 *              such operation, and exact semantics depends on the <tt>URL</tt>
527 *              implementations. In case of {@link HttpURLConnection HTTP URLs},
528 *              this will perform HTTP POST.</td>
529 *      </tr><tr>
530 *          <td>{@link URI}</td>
531 *          <td>The URI is {@link URI#toURL() turned into URL} and then
532 *              follows the handling of <tt>URL</tt>. See above.</td>
533 *      </tr><tr>
534 *          <td>{@link String}</td>
535 *          <td>The string is first interpreted as an absolute <tt>URI</tt>.
536 *              If it's not {@link URI#isAbsolute() a valid absolute URI},
537 *              then it's interpreted as a <tt>File</tt></td>
538 *      </tr><tr>
539 *          <td>{@link OutputStream}</td>
540 *          <td>The XML will be sent to the given {@link OutputStream}.
541 *              Upon a successful completion, the stream will be closed by this method.</td>
542 *      </tr><tr>
543 *          <td>{@link Writer}</td>
544 *          <td>The XML will be sent as a character stream to the given {@link Writer}.
545 *              Upon a successful completion, the stream will be closed by this method.</td>
546 *      </tr><tr>
547 *          <td>{@link Result}</td>
548 *          <td>The XML will be sent to the {@link Result} object.</td>
549 *      </tr></table>
550 *
551 * @throws DataBindingException
552 *     If the operation fails, such as due to I/O error, unbindable classes.
553 */
554 private static void _marshal( Object jaxbObject, Object xml ) {
555     try {
556         JAXBContext context;
557
558         if(jaxbObject instanceof JAXBElement) {
559             context = getContext(((JAXBElement<?>)jaxbObject).getDeclaredType());
560         } else {
561             Class<?> clazz = jaxbObject.getClass();
562             XmlRootElement r = clazz.getAnnotation(XmlRootElement.class);
563             context = getContext(clazz);
564             if(r==null) {
565                 // we need to infer the name
566                 jaxbObject = new JAXBElement(new QName(inferName(clazz)),clazz,jaxbObject);
567             }
568         }
569
570         Marshaller m = context.createMarshaller();

```

```

571         m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,true);
572         m.marshal(jaxbObject, toResult(xml));
573     } catch (JAXBException e) {
574         throw new DataBindingException(e);
575     } catch (IOException e) {
576         throw new DataBindingException(e);
577     }
578 }
579
580 private static String inferName(Class clazz) {
581     return Introspector.decapitalize(clazz.getSimpleName());
582 }
583
584 /**
585  * Creates {@link Result} from various XML representation.
586  * See {@link #_marshal(Object,Object)} for the conversion rules.
587  */
588 private static Result toResult(Object xml) throws IOException {
589     if(xml==null)
590         throw new IllegalArgumentException("no XML is given");
591
592     if (xml instanceof String) {
593         try {
594             xml=new URI((String)xml);
595         } catch (URISyntaxException e) {
596             xml=new File((String)xml);
597         }
598     }
599     if (xml instanceof File) {
600         File file = (File) xml;
601         return new StreamResult(file);
602     }
603     if (xml instanceof URI) {
604         URI uri = (URI) xml;
605         xml=uri.toURL();
606     }
607     if (xml instanceof URL) {
608         URL url = (URL) xml;
609         URLConnection con = url.openConnection();
610         con.setDoOutput(true);
611         con.setDoInput(false);
612         con.connect();
613         return new StreamResult(con.getOutputStream());
614     }
615     if (xml instanceof OutputStream) {
616         OutputStream os = (OutputStream) xml;
617         return new StreamResult(os);
618     }
619     if (xml instanceof Writer) {
620         Writer w = (Writer)xml;
621         return new StreamResult(w);
622     }
623     if (xml instanceof Result) {

```



```

624         return (Result) xml;
625     }
626     throw new IllegalArgumentException("I don't understand how to handle "+xml.getClass());
627 }
628
629 }

```

## 2.10 JAXBContext.java

Secuencia 12: javax/xml/bind/JAXBContext.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import org.w3c.dom.Node;
29
30 import java.util.Collections;
31 import java.util.Map;
32 import java.util.Properties;
33 import java.io.IOException;
34 import java.io.InputStream;
35
36 /**
37  * <p>
38  * The <tt>JAXBContext</tt> class provides the client's entry point to the
39  * JAXB API. It provides an abstraction for managing the XML/Java binding
40  * information necessary to implement the JAXB binding framework operations:
41  * unmarshal, marshal and validate.
42  *
43  * <p>A client application normally obtains new instances of this class using

```

```

44 * one of these two styles for newInstance methods, although there are other
45 * specialized forms of the method available:
46 *
47 * <ul>
48 *   <li>{@link #newInstance(String,ClassLoader) JAXBContext.newInstance( "com.acme.foo:com.acme.
49 *     bar" )} <br/>
50 *   The JAXBContext instance is initialized from a list of colon
51 *   separated Java package names. Each java package contains
52 *   JAXB mapped classes, schema-derived classes and/or user annotated
53 *   classes. Additionally, the java package may contain JAXB package annotations
54 *   that must be processed. (see JLS, Section 7.4.1 "Named Packages").
55 *   </li>
56 *   <li>{@link #newInstance(Class...) JAXBContext.newInstance( com.acme.foo.Foo.class )} <br/>
57 *   The JAXBContext instance is initialized with class(es)
58 *   passed as parameter(s) and classes that are statically reachable from
59 *   these class(es). See {@link #newInstance(Class...)} for details.
60 *   </li>
61 * </ul>
62 *
63 * <p>
64 * <i><B>SPEC REQUIREMENT:</B> the provider must supply an implementation
65 * class containing the following method signatures:</i>
66 *
67 * <pre>
68 * public static JAXBContext createContext( String contextPath, ClassLoader classLoader, Map<
69 *   String,Object> properties ) throws JAXBException
70 * public static JAXBContext createContext( Class[] classes, Map<String,Object> properties )
71 *   throws JAXBException
72 * </pre>
73 *
74 * <p><i>
75 * The following JAXB 1.0 requirement is only required for schema to
76 * java interface/implementation binding. It does not apply to JAXB annotated
77 * classes. JAXB Providers must generate a <tt>jaxb.properties</tt> file in
78 * each package containing schema derived classes. The property file must
79 * contain a property named <tt>javax.xml.bind.context.factory</tt> whose
80 * value is the name of the class that implements the <tt>createContext</tt>
81 * APIs.</i>
82 *
83 * <p><i>
84 * The class supplied by the provider does not have to be assignable to
85 * <tt>javax.xml.bind.JAXBContext</tt>, it simply has to provide a class that
86 * implements the <tt>createContext</tt> APIs.</i>
87 *
88 * <p><i>
89 * In addition, the provider must call the
90 * {@link DatatypeConverter#setDatatypeConverter(DatatypeConverterInterface)
91 * DatatypeConverter.setDatatypeConverter} api prior to any client
92 * invocations of the marshal and unmarshal methods. This is necessary to
93 * configure the datatype converter that will be used during these operations.</i>
94 *
95 * <a name="Unmarshalling"></a>
96 * <h3>Unmarshalling</h3>

```

```

94 * <p>
95 * The {@link Unmarshaller} class provides the client application the ability
96 * to convert XML data into a tree of Java content objects.
97 * The unmarshal method allows for
98 * any global XML element declared in the schema to be unmarshalled as
99 * the root of an instance document.
100 * Additionally, the unmarshal method allows for an unrecognized root element that
101 * has an xsi:type attribute's value that references a type definition declared in
102 * the schema to be unmarshalled as the root of an instance document.
103 * The <tt>JAXBContext</tt> object
104 * allows the merging of global elements and type definitions across a set of schemas (listed
105 * in the <tt>contextPath</tt>). Since each schema in the schema set can belong
106 * to distinct namespaces, the unification of schemas to an unmarshalling
107 * context should be namespace independent. This means that a client
108 * application is able to unmarshal XML documents that are instances of
109 * any of the schemas listed in the <tt>contextPath</tt>. For example:
110 *
111 * <pre>
112 *     JAXBContext jc = JAXBContext.newInstance( "com.acme.foo:com.acme.bar" );
113 *     Unmarshaller u = jc.createUnmarshaller();
114 *     FooObject fooObj = (FooObject)u.unmarshal( new File( "foo.xml" ) ); // ok
115 *     BarObject barObj = (BarObject)u.unmarshal( new File( "bar.xml" ) ); // ok
116 *     BazObject bazObj = (BazObject)u.unmarshal( new File( "baz.xml" ) ); // error, "com.acme.
        baz" not in contextPath
117 * </pre>
118 *
119 * <p>
120 * The client application may also generate Java content trees explicitly rather
121 * than unmarshalling existing XML data. For all JAXB-annotated value classes,
122 * an application can create content using constructors.
123 * For schema-derived interface/implementation classes and for the
124 * creation of elements that are not bound to a JAXB-annotated
125 * class, an application needs to have access and knowledge about each of
126 * the schema derived <tt>ObjectFactory</tt> classes that exist in each of
127 * java packages contained in the <tt>contextPath</tt>. For each schema
128 * derived java class, there is a static factory method that produces objects
129 * of that type. For example,
130 * assume that after compiling a schema, you have a package <tt>com.acme.foo</tt>
131 * that contains a schema derived interface named <tt>PurchaseOrder</tt>. In
132 * order to create objects of that type, the client application would use the
133 * factory method like this:
134 *
135 * <pre>
136 *     com.acme.foo.PurchaseOrder po =
137 *         com.acme.foo.ObjectFactory.createPurchaseOrder();
138 * </pre>
139 *
140 * <p>
141 * Once the client application has an instance of the the schema derived object,
142 * it can use the mutator methods to set content on it.
143 *
144 * <p>
145 * For more information on the generated <tt>ObjectFactory</tt> classes, see

```

```

146 * Section 4.2 <i>Java Package</i> of the specification.
147 *
148 * <p>
149 * <i><B>SPEC REQUIREMENT:</B></i> the provider must generate a class in each
150 * package that contains all of the necessary object factory methods for that
151 * package named ObjectFactory as well as the static
152 * <tt>newInstance( javaContentInterface )</tt> method</i>
153 *
154 * <h3>Marshalling</h3>
155 * <p>
156 * The {@link Marshaller} class provides the client application the ability
157 * to convert a Java content tree back into XML data. There is no difference
158 * between marshalling a content tree that is created manually using the factory
159 * methods and marshalling a content tree that is the result an <tt>unmarshal
160 * </tt> operation. Clients can marshal a java content tree back to XML data
161 * to a <tt>java.io.OutputStream</tt> or a <tt>java.io.Writer</tt>. The
162 * marshalling process can alternatively produce SAX2 event streams to a
163 * registered <tt>ContentHandler</tt> or produce a DOM Node object.
164 * Client applications have control over the output encoding as well as
165 * whether or not to marshal the XML data as a complete document or
166 * as a fragment.
167 *
168 * <p>
169 * Here is a simple example that unmarshals an XML document and then marshals
170 * it back out:
171 *
172 * <pre>
173 *     JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
174 *
175 *     // unmarshal from foo.xml
176 *     Unmarshaller u = jc.createUnmarshaller();
177 *     FooObject fooObj = (FooObject)u.unmarshal( new File( "foo.xml" ) );
178 *
179 *     // marshal to System.out
180 *     Marshaller m = jc.createMarshaller();
181 *     m.marshal( fooObj, System.out );
182 * </pre>
183 *
184 *
185 * <h3>Validation</h3>
186 * <p>
187 * Validation has been changed significantly since JAXB 1.0. The {@link Validator}
188 * class has been deprecated and made optional. This means that you are advised
189 * not to use this class and, in fact, it may not even be available depending on
190 * your JAXB provider. JAXB 1.0 client applications that rely on <tt>Validator</tt>
191 * will still work properly when deployed with the JAXB 1.0 runtime system.
192 *
193 * In JAXB 2.0, the {@link Unmarshaller} has included convenience methods that expose
194 * the JAXP 1.3 {@link javax.xml.validation} framework. Please refer to the
195 * {@link Unmarshaller#setSchema(javax.xml.validation.Schema)} API for more
196 * information.
197 *
198 *

```

```

199 * <h3>JAXB Runtime Binding Framework Compatibility</h3>
200 * <p>
201 * The following JAXB 1.0 restriction only applies to binding schema to
202 * interfaces/implementation classes.
203 * Since this binding does not require a common runtime system, a JAXB
204 * client application must not attempt to mix runtime objects (<tt>JAXBContext,
205 * Marshaller</tt>, etc. ) from different providers. This does not
206 * mean that the client application isn't portable, it simply means that a
207 * client has to use a runtime system provided by the same provider that was
208 * used to compile the schema.
209 *
210 *
211 * <h3>Discovery of JAXB implementation</h3>
212 * <p>
213 * When one of the <tt>newInstance</tt> methods is called, a JAXB implementation is discovered
214 * by the following steps.
215 *
216 * <ol>
217 * <li>
218 * For each package/class explicitly passed in to the {@link #newInstance} method, in the order
219 * they are specified,
220 * <tt>jaxb.properties</tt> file is looked up in its package, by using the associated classloader &
221 * mdash;
222 * this is {@link Class#getClassLoader() the owner class loader} for a {@link Class} argument, and
223 * for a package
224 * the specified {@link ClassLoader}.
225 *
226 * <p>
227 * If such a file is discovered, it is {@link Properties#load(InputStream) loaded} as a property
228 * file, and
229 * the value of the {@link #JAXB_CONTEXT_FACTORY} key will be assumed to be the provider factory
230 * class.
231 * This class is then loaded by the associated classloader discussed above.
232 *
233 * <p>
234 * This phase of the look up allows some packages to force the use of a certain JAXB implementation
235 * .
236 * (For example, perhaps the schema compiler has generated some vendor extension in the code.)
237 *
238 * <li>
239 * If the system property {@link #JAXB_CONTEXT_FACTORY} exists, then its value is assumed to be the
240 * provider
241 * factory class. This phase of the look up enables per-JVM override of the JAXB implementation.
242 *
243 * <li>
244 * Look for <tt>META-INF/services/javax.xml.bind.JAXBContext</tt> file in the associated
245 * classloader.
246 * This file follows the standard service descriptor convention, and if such a file exists, its
247 * content
248 * is assumed to be the provider factory class. This phase of the look up is for automatic
249 * discovery.
250 * It allows users to just put a JAXB implementation in a classpath and use it without any further
251 * configuration.

```

```

241 *
242 * <li>
243 * Finally, if all the steps above fail, then the rest of the look up is unspecified. That said,
244 * the recommended behavior is to simply look for some hard-coded platform default JAXB
    implementation.
245 * This phase of the look up is so that JavaSE can have its own JAXB implementation as the last
    resort.
246 * </ol>
247 *
248 * <p>
249 * Once the provider factory class is discovered, its
250 * <tt>public static JAXBContext createContext(String,ClassLoader,Map)</tt> method
251 * (see {@link #newInstance(String, ClassLoader, Map)} for the parameter semantics.)
252 * or <tt>public static JAXBContext createContet(Class[],Map)</tt> method
253 * (see {@link #newInstance(Class[], Map)} for the parameter semantics) are invoked
254 * to create a {@link JAXBContext}.
255 *
256 * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
    Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
257 * @see Marshaller
258 * @see Unmarshaller
259 * @see S 7.4.1 "Named Packages" in Java Language Specification</a>
260 * @since JAXB1.0
261 */
262 public abstract class JAXBContext {
263
264     /**
265      * The name of the property that contains the name of the class capable
266      * of creating new <tt>JAXBContext</tt> objects.
267      */
268     public static final String JAXB_CONTEXT_FACTORY =
269         "javax.xml.bind.context.factory";
270
271
272     protected JAXBContext() {
273     }
274
275
276     /**
277      * <p>
278      * Obtain a new instance of a <tt>JAXBContext</tt> class.
279      *
280      * <p>
281      * This is a convenience method to invoke the
282      * {@link #newInstance(String,ClassLoader)} method with
283      * the context class loader of the current thread.
284      *
285      * @throws JAXBException if an error was encountered while creating the
286      *      <tt>JAXBContext</tt> such as
287      *      <ol>
288      *      <li>failure to locate either ObjectFactory.class or jaxb.index in the packages</li>
289      *      <li>an ambiguity among global elements contained in the contextPath</li>
290      *      <li>failure to locate a value for the context factory provider property</li>

```

```

291     * <li>mixing schema derived packages from different providers on the same contextPath</li>
292     * </ol>
293     */
294     public static JAXBContext newInstance( String contextPath )
295         throws JAXBException {
296
297         //return newInstance( contextPath, JAXBContext.class.getClassLoader() );
298         return newInstance( contextPath, getContextClassLoader());
299     }
300
301     /**
302     * <p>
303     * Obtain a new instance of a <tt>JAXBContext</tt> class.
304     *
305     * <p>
306     * The client application must supply a context path which is a list of
307     * colon (':', \u005Cu003A) separated java package names that contain
308     * schema-derived classes and/or fully qualified JAXB-annotated classes.
309     * Schema-derived
310     * code is registered with the JAXBContext by the
311     * ObjectFactory.class generated per package.
312     * Alternatively than being listed in the context path, programmer
313     * annotated JAXB mapped classes can be listed in a
314     * <tt>jaxb.index</tt> resource file, format described below.
315     * Note that a java package can contain both schema-derived classes and
316     * user annotated JAXB classes. Additionally, the java package may
317     * contain JAXB package annotations that must be processed. (see JLS,
318     * Section 7.4.1 "Named Packages").
319     * </p>
320     *
321     * <p>
322     * Every package listed on the contextPath must meet <b>one or both</b> of the
323     * following conditions otherwise a <tt>JAXBException</tt> will be thrown:
324     * </p>
325     * <ol>
326     * <li>it must contain ObjectFactory.class</li>
327     * <li>it must contain jaxb.index</li>
328     * </ol>
329     *
330     * <p>
331     * <b>Format for jaxb.index</b>
332     * <p>
333     * The file contains a newline-separated list of class names.
334     * Space and tab characters, as well as blank
335     * lines, are ignored. The comment character
336     * is '#' (0x23); on each line all characters following the first comment
337     * character are ignored. The file must be encoded in UTF-8. Classes that
338     * are reachable, as defined in {@link #newInstance(Class...)}, from the
339     * listed classes are also registered with JAXBContext.
340     * <p>
341     * Constraints on class name occuring in a <tt>jaxb.index</tt> file are:
342     * <ul>
343     * <li>Must not end with ".class".</li>

```

```

344 * <li>Class names are resolved relative to package containing
345 * <tt>jaxb.index</tt> file. Only classes occurring directly in package
346 * containing <tt>jaxb.index</tt> file are allowed.</li>
347 * <li>Fully qualified class names are not allowed.
348 * A qualified class name,relative to current package,
349 * is only allowed to specify a nested or inner class.</li>
350 * </ul>
351 *
352 * <p>
353 * To maintain compatibility with JAXB 1.0 schema to java
354 * interface/implementation binding, enabled by schema customization
355 * <tt><?xml:namespace prefix="j" namespace="http://www.w3.org/2001/XMLSchema-instance" type="boolean" value="false"/></tt>,
356 * the JAXB provider will ensure that each package on the context path
357 * has a <tt>jaxb.properties</tt> file which contains a value for the
358 * <tt>javax.xml.bind.context.factory</tt> property and that all values
359 * resolve to the same provider. This requirement does not apply to
360 * JAXB annotated classes.
361 *
362 * <p>
363 * If there are any global XML element name collisions across the various
364 * packages listed on the <tt>contextPath</tt>, a <tt>JAXBException</tt>
365 * will be thrown.
366 *
367 * <p>
368 * Mixing generated interface/impl bindings from multiple JAXB Providers
369 * in the same context path may result in a <tt>JAXBException</tt>
370 * being thrown.
371 *
372 * <p>
373 * The steps involved in discovering the JAXB implementation is discussed in the class javadoc.
374 *
375 * @param contextPath list of java package names that contain schema
376 * derived class and/or java to schema (JAXB-annotated)
377 * mapped classes
378 * @param classLoader
379 * This class loader will be used to locate the implementation
380 * classes.
381 *
382 * @return a new instance of a <tt>JAXBContext</tt>
383 * @throws JAXBException if an error was encountered while creating the
384 * <tt>JAXBContext</tt> such as
385 * <ol>
386 * <li>failure to locate either ObjectFactory.class or jaxb.index in the packages</li>
387 * <li>an ambiguity among global elements contained in the contextPath</li>
388 * <li>failure to locate a value for the context factory provider property</li>
389 * <li>mixing schema derived packages from different providers on the same contextPath</li>
390 * </ol>
391 */
392 public static JAXBContext newInstance( String contextPath, ClassLoader classLoader ) throws
    JAXBException {
393
394     return newInstance(contextPath,classLoader,Collections.<String,Object>emptyMap());
395 }

```



```

396
397 /**
398  * <p>
399  * Obtain a new instance of a <tt>JAXBContext</tt> class.
400  *
401  * <p>
402  * This is mostly the same as {@link JAXBContext#newInstance(String, ClassLoader)},
403  * but this version allows you to pass in provider-specific properties to configure
404  * the instantiation of {@link JAXBContext}.
405  *
406  * <p>
407  * The interpretation of properties is up to implementations. Implementations should
408  * throw <tt>JAXBException</tt> if it finds properties that it doesn't understand.
409  *
410  * @param contextPath list of java package names that contain schema derived classes
411  * @param classLoader
412  *     This class loader will be used to locate the implementation classes.
413  * @param properties
414  *     provider-specific properties. Can be null, which means the same thing as passing
415  *     in an empty map.
416  *
417  * @return a new instance of a <tt>JAXBContext</tt>
418  * @throws JAXBException if an error was encountered while creating the
419  *     <tt>JAXBContext</tt> such as
420  * <ol>
421  * <li>failure to locate either ObjectFactory.class or jaxb.index in the packages</li>
422  * <li>an ambiguity among global elements contained in the contextPath</li>
423  * <li>failure to locate a value for the context factory provider property</li>
424  * <li>mixing schema derived packages from different providers on the same contextPath</li>
425  * </ol>
426  * @since JAXB2.0
427  */
428 public static JAXBContext newInstance( String contextPath, ClassLoader classLoader, Map<String
429     ,?> properties )
430     throws JAXBException {
431
432     return ContextFinder.find(
433         /* The default property name according to the JAXB spec */
434         JAXB_CONTEXT_FACTORY,
435
436         /* the context path supplied by the client app */
437         contextPath,
438
439         /* class loader to be used */
440         classLoader,
441         properties );
442
443 // TODO: resurrect this once we introduce external annotations
444 // /**
445 //  * <p>
446 //  * Obtain a new instance of a <tt>JAXBContext</tt> class.
447 //  *

```

```

448 //      * <p>
449 //      * The client application must supply a list of classes that the new
450 //      * context object needs to recognize.
451 //      *
452 //      * Not only the new context will recognize all the classes specified,
453 //      * but it will also recognize any classes that are directly/indirectly
454 //      * referenced statically from the specified classes.
455 //      *
456 //      * For example, in the following Java code, if you do
457 //      * <tt>newInstance(Foo.class)</tt>, the newly created {@link JAXBContext}
458 //      * will recognize both <tt>Foo</tt> and <tt>Bar</tt>, but not <tt>Zot</tt>:
459 //      * <pre>
460 //      * class Foo {
461 //      *     Bar b;
462 //      * }
463 //      * class Bar { int x; }
464 //      * class Zot extends Bar { int y; }
465 //      * </pre>
466 //      *
467 //      * Therefore, a typical client application only needs to specify the
468 //      * top-level classes, but it needs to be careful.
469 //      *
470 //      * TODO: if we are to define other mechanisms, refer to them.
471 //      *
472 //      * @param externalBindings
473 //      *      list of external binding files. Can be null or empty if none is used.
474 //      *      when specified, those files determine how the classes are bound.
475 //      *
476 //      * @param classesToBeBound
477 //      *      list of java classes to be recognized by the new {@link JAXBContext}.
478 //      *      Can be empty, in which case a {@link JAXBContext} that only knows about
479 //      *      spec-defined classes will be returned.
480 //      *
481 //      * @return
482 //      *      A new instance of a <tt>JAXBContext</tt>. Always non-null valid object.
483 //      *
484 //      * @throws JAXBException
485 //      *      if an error was encountered while creating the
486 //      *      <tt>JAXBContext</tt>, such as (but not limited to):
487 //      *      <ol>
488 //      *      <li>No JAXB implementation was discovered
489 //      *      <li>Classes use JAXB annotations incorrectly
490 //      *      <li>Classes have colliding annotations (i.e., two classes with the same type name)
491 //      *      <li>Specified external bindings are incorrect
492 //      *      <li>The JAXB implementation was unable to locate
493 //      *          provider-specific out-of-band information (such as additional
494 //      *          files generated at the development time.)
495 //      *      </ol>
496 //      *
497 //      * @throws IllegalArgumentException
498 //      *      if the parameter contains {@code null} (i.e., {@code newInstance(null)};})
499 //      *
500 //      * @since JAXB2.0

```

```

501 //      */
502 //      public static JAXBContext newInstance( Source[] externalBindings, Class... classesToBeBound )
503 //          throws JAXBException {
504 //
505 //          // empty class list is not an error, because the context will still include
506 //          // spec-specified classes like String and Integer.
507 //          // if(classesToBeBound.length==0)
508 //          //      throw new IllegalArgumentException();
509 //
510 //          // but it is an error to have nulls in it.
511 //          for( int i=classesToBeBound.length-1; i>=0; i-- )
512 //              if(classesToBeBound[i]==null)
513 //                  throw new IllegalArgumentException();
514 //
515 //          return ContextFinder.find(externalBindings,classesToBeBound);
516 //      }
517
518 /**
519  * <p>
520  * Obtain a new instance of a <tt>JAXBContext</tt> class.
521  *
522  * <p>
523  * The client application must supply a list of classes that the new
524  * context object needs to recognize.
525  *
526  * Not only the new context will recognize all the classes specified,
527  * but it will also recognize any classes that are directly/indirectly
528  * referenced statically from the specified classes. Subclasses of
529  * referenced classes nor <tt>@XmlTransient</tt> referenced classes
530  * are not registered with JAXBContext.
531  *
532  * For example, in the following Java code, if you do
533  * <tt>newInstance(Foo.class)</tt>, the newly created {@link JAXBContext}
534  * will recognize both <tt>Foo</tt> and <tt>Bar</tt>, but not <tt>Zot</tt> or <tt>FooBar</tt>:
535  * <pre>
536  * class Foo {
537  *     @XmlTransient FooBar c;
538  *     Bar b;
539  * }
540  * class Bar { int x; }
541  * class Zot extends Bar { int y; }
542  * class FooBar { }
543  * </pre>
544  *
545  * Therefore, a typical client application only needs to specify the
546  * top-level classes, but it needs to be careful.
547  *
548  * <p>
549  * Note that for each java package registered with JAXBContext,
550  * when the optional package annotations exist, they must be processed.
551  * (see JLS, Section 7.4.1 "Named Packages").
552  *
553  * <p>

```

```

554 * The steps involved in discovering the JAXB implementation is discussed in the class javadoc.
555 *
556 * @param classesToBeBound
557 *     list of java classes to be recognized by the new {@link JAXBContext}.
558 *     Can be empty, in which case a {@link JAXBContext} that only knows about
559 *     spec-defined classes will be returned.
560 *
561 * @return
562 *     A new instance of a <tt>JAXBContext</tt>. Always non-null valid object.
563 *
564 * @throws JAXBException
565 *     if an error was encountered while creating the
566 *     <tt>JAXBContext</tt>, such as (but not limited to):
567 *     <ol>
568 *     <li>No JAXB implementation was discovered
569 *     <li>Classes use JAXB annotations incorrectly
570 *     <li>Classes have colliding annotations (i.e., two classes with the same type name)
571 *     <li>The JAXB implementation was unable to locate
572 *         provider-specific out-of-band information (such as additional
573 *         files generated at the development time.)
574 *     </ol>
575 *
576 * @throws IllegalArgumentException
577 *     if the parameter contains {@code null} (i.e., {@code newInstance(null)};})
578 *
579 * @since JAXB2.0
580 */
581 public static JAXBContext newInstance( Class... classesToBeBound )
582     throws JAXBException {
583
584     return newInstance(classesToBeBound,Collections.<String,Object>emptyMap());
585 }
586
587 /**
588 * <p>
589 * Obtain a new instance of a <tt>JAXBContext</tt> class.
590 *
591 * <p>
592 * An overloading of {@link JAXBContext#newInstance(Class...)}
593 * to configure 'properties' for this instantiation of {@link JAXBContext}.
594 *
595 * <p>
596 * The interpretation of properties is up to implementations. Implementations should
597 * throw <tt>JAXBException</tt> if it finds properties that it doesn't understand.
598 *
599 * @param classesToBeBound
600 *     list of java classes to be recognized by the new {@link JAXBContext}.
601 *     Can be empty, in which case a {@link JAXBContext} that only knows about
602 *     spec-defined classes will be returned.
603 * @param properties
604 *     provider-specific properties. Can be null, which means the same thing as passing
605 *     in an empty map.
606 *

```

```

607 * @return
608 *     A new instance of a <tt>JAXBContext</tt>. Always non-null valid object.
609 *
610 * @throws JAXBException
611 *     if an error was encountered while creating the
612 *     <tt>JAXBContext</tt>, such as (but not limited to):
613 * <ol>
614 * <li>No JAXB implementation was discovered
615 * <li>Classes use JAXB annotations incorrectly
616 * <li>Classes have colliding annotations (i.e., two classes with the same type name)
617 * <li>The JAXB implementation was unable to locate
618 *     provider-specific out-of-band information (such as additional
619 *     files generated at the development time.)
620 * </ol>
621 *
622 * @throws IllegalArgumentException
623 *     if the parameter contains {@code null} (i.e., {@code newInstance(null,someMap)};})
624 *
625 * @since JAXB2.0
626 */
627 public static JAXBContext newInstance( Class[] classesToBeBound, Map<String,?> properties )
628     throws JAXBException {
629
630     if (classesToBeBound == null) {
631         throw new IllegalArgumentException();
632     }
633
634     // but it is an error to have nulls in it.
635     for (int i = classesToBeBound.length - 1; i >= 0; i--) {
636         if (classesToBeBound[i] == null) {
637             throw new IllegalArgumentException();
638         }
639     }
640
641     return ContextFinder.find(classesToBeBound,properties);
642 }
643
644 /**
645 * Create an <tt>Unmarshaller</tt> object that can be used to convert XML
646 * data into a java content tree.
647 *
648 * @return an <tt>Unmarshaller</tt> object
649 *
650 * @throws JAXBException if an error was encountered while creating the
651 *     <tt>Unmarshaller</tt> object
652 */
653 public abstract Unmarshaller createUnmarshaller() throws JAXBException;
654
655
656 /**
657 * Create a <tt>Marshaller</tt> object that can be used to convert a
658 * java content tree into XML data.
659 *

```

```

660     * @return a <tt>Marshaller</tt> object
661     *
662     * @throws JAXBException if an error was encountered while creating the
663     *         <tt>Marshaller</tt> object
664     */
665     public abstract Marshaller createMarshaller() throws JAXBException;
666
667
668     /**
669     * {@link Validator} has been made optional and deprecated in JAXB 2.0. Please
670     * refer to the javadoc for {@link Validator} for more detail.
671     * <p>
672     * Create a <tt>Validator</tt> object that can be used to validate a
673     * java content tree against its source schema.
674     *
675     * @return a <tt>Validator</tt> object
676     *
677     * @throws JAXBException if an error was encountered while creating the
678     *         <tt>Validator</tt> object
679     * @deprecated since JAXB2.0
680     */
681     public abstract Validator createValidator() throws JAXBException;
682
683     /**
684     * Creates a <tt>Binder</tt> object that can be used for
685     * associative/in-place unmarshalling/marshalling.
686     *
687     * @param domType select the DOM API to use by passing in its DOM Node class.
688     *
689     * @return always a new valid <tt>Binder</tt> object.
690     *
691     * @throws UnsupportedOperationException
692     *         if DOM API corresponding to <tt>domType</tt> is not supported by
693     *         the implementation.
694     *
695     * @since JAXB2.0
696     */
697     public <T> Binder<T> createBinder(Class<T> domType) {
698         // to make JAXB 1.0 implementations work, this method must not be
699         // abstract
700         throw new UnsupportedOperationException();
701     }
702
703     /**
704     * Creates a <tt>Binder</tt> for W3C DOM.
705     *
706     * @return always a new valid <tt>Binder</tt> object.
707     *
708     * @since JAXB2.0
709     */
710     public Binder<Node> createBinder() {
711         return createBinder(Node.class);
712     }

```

```

713
714 /**
715  * Creates a <tt>JAXBIntrospector</tt> object that can be used to
716  * introspect JAXB objects.
717  *
718  * @return
719  *     always return a non-null valid <tt>JAXBIntrospector</tt> object.
720  *
721  * @throws UnsupportedOperationException
722  *     Calling this method on JAXB 1.0 implementations will throw
723  *     an UnsupportedOperationException.
724  *
725  * @since JAXB2.0
726  */
727 public JAXBIntrospector createJAXBIntrospector() {
728     // to make JAXB 1.0 implementations work, this method must not be
729     // abstract
730     throw new UnsupportedOperationException();
731 }
732
733 /**
734  * Generates the schema documents for this context.
735  *
736  * @param outputResolver
737  *     this object controls the output to which schemas
738  *     will be sent.
739  *
740  * @throws IOException
741  *     if {@link SchemaOutputResolver} throws an {@link IOException}.
742  *
743  * @throws UnsupportedOperationException
744  *     Calling this method on JAXB 1.0 implementations will throw
745  *     an UnsupportedOperationException.
746  *
747  * @since JAXB 2.0
748  */
749 public void generateSchema(SchemaOutputResolver outputResolver) throws IOException {
750     // to make JAXB 1.0 implementations work, this method must not be
751     // abstract
752     throw new UnsupportedOperationException();
753 }
754
755 private static ClassLoader getContextClassLoader() {
756     if (System.getSecurityManager() == null) {
757         return Thread.currentThread().getContextClassLoader();
758     } else {
759         return (ClassLoader) java.security.AccessController.doPrivileged(
760             new java.security.PrivilegedAction() {
761                 public java.lang.Object run() {
762                     return Thread.currentThread().getContextClassLoader();
763                 }
764             });
765     }

```

```

766     }
767
768 }

```

## 2.11 JAXBElement.java

Secuencia 13: javax/xml/bind/JAXBElement.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import javax.xml.namespace.QName;
29 import java.io.Serializable;
30
31 /**
32  * <p>JAXB representation of an Xml Element.</p>
33  *
34  * <p>This class represents information about an Xml Element from both the element
35  * declaration within a schema and the element instance value within an xml document
36  * with the following properties
37  * <ul>
38  *   <li>element's xml tag <b><tt>name</tt></b></li>
39  *   <li><b><tt>value</tt></b> represents the element instance's attribute(s) and content model</li>
40  *   <li>element declaration's <b><tt>declaredType</tt></b> (<tt>xs:element @type</tt> attribute)</li>
41  *   <li><b><tt>scope</tt></b> of element declaration</li>
42  *   <li>boolean <b><tt>nil</tt></b> property. (element instance's <tt><b>xsi:nil</b></tt> attribute)</li>
43  * </ul>

```



```

44  *
45  * <p>The <tt>declaredType</tt> and <tt>scope</tt> property are the
46  * JAXB class binding for the xml type definition.
47  * </p>
48  *
49  * <p><b><tt>Scope</tt></b> is either {@link GlobalScope} or the Java class representing the
50  * complex type definition containing the schema element declaration.
51  * </p>
52  *
53  * <p>There is a property constraint that if <b><tt>value</tt></b> is <tt>null</tt>,
54  * then <tt>nil</tt> must be <tt>true</tt>. The converse is not true to enable
55  * representing a nil element with attribute(s). If <tt>nil</tt> is true, it is possible
56  * that <tt>value</tt> is non-null so it can hold the value of the attributes
57  * associated with a nil element.
58  * </p>
59  *
60  * @author Kohsuke Kawaguchi, Joe Fialli
61  * @since JAXB 2.0
62  */
63
64  public class JAXBElement<T> implements Serializable {
65
66      /** xml element tag name */
67      final protected QName name;
68
69      /** Java datatype binding for xml element declaration's type. */
70      final protected Class<T> declaredType;
71
72      /** Scope of xml element declaration representing this xml element instance.
73       * Can be one of the following values:
74       * - {@link GlobalScope} for global xml element declaration.
75       * - local element declaration has a scope set to the Java class
76       *   representation of complex type definition containing
77       *   xml element declaration.
78       */
79      final protected Class scope;
80
81      /** xml element value.
82       * Represents content model and attributes of an xml element instance. */
83      protected T value;
84
85      /** true iff the xml element instance has xsi:nil="true". */
86      protected boolean nil = false;
87
88      /**
89       * Designates global scope for an xml element.
90       */
91      public static final class GlobalScope {}
92
93      /**
94       * <p>Construct an xml element instance.</p>
95       *
96       * @param name      Java binding of xml element tag name

```

```

97  * @param declaredType Java binding of xml element declaration's type
98  * @param scope
99  *     Java binding of scope of xml element declaration.
100 *     Passing null is the same as passing <tt>GlobalScope.class</tt>
101 * @param value
102 *     Java instance representing xml element's value.
103 * @see #getScope()
104 * @see #isTypeSubstituted()
105 */
106 public JAXBElement(QName name,
107                   Class<T> declaredType,
108                   Class scope,
109                   T value) {
110     if(declaredType==null || name==null)
111         throw new IllegalArgumentException();
112     this.declaredType = declaredType;
113     if(scope==null)     scope = GlobalScope.class;
114     this.scope = scope;
115     this.name = name;
116     setValue(value);
117 }
118
119 /**
120  * Construct an xml element instance.
121  *
122  * This is just a convenience method for <tt>new JAXBElement(name,declaredType,GlobalScope.
123  * class,value)</tt>
124  */
125 public JAXBElement(QName name, Class<T> declaredType, T value ) {
126     this(name,declaredType,GlobalScope.class,value);
127 }
128
129 /**
130  * Returns the Java binding of the xml element declaration's type attribute.
131  */
132 public Class<T> getDeclaredType() {
133     return declaredType;
134 }
135
136 /**
137  * Returns the xml element tag name.
138  */
139 public QName getName() {
140     return name;
141 }
142
143 /**
144  * <p>Set the content model and attributes of this xml element.</p>
145  *
146  * <p>When this property is set to <tt>null</tt>, <tt>isNil()</tt> must by <tt>true</tt>.
147  * Details of constraint are described at {@link #isNil()}.</p>
148  *
149  * @see #isTypeSubstituted()

```

```

149     */
150     public void setValue(T t) {
151         this.value = t;
152     }
153
154     /**
155      * <p>Return the content model and attribute values for this element.</p>
156      *
157      * <p>See {@link #isNil()} for a description of a property constraint when
158      * this value is <tt>null</tt></p>
159      */
160     public T getValue() {
161         return value;
162     }
163
164     /**
165      * Returns scope of xml element declaration.
166      *
167      * @see #isGlobalScope()
168      * @return <tt>GlobalScope.class</tt> if this element is of global scope.
169      */
170     public Class getScope() {
171         return scope;
172     }
173
174     /**
175      * <p>Returns <tt>true</tt> iff this element instance content model
176      * is nil.</p>
177      *
178      * <p>This property always returns <tt>true</tt> when {@link #getValue()} is null.
179      * Note that the converse is not true, when this property is <tt>true</tt>,
180      * {@link #getValue()} can contain a non-null value for attribute(s). It is
181      * valid for a nil xml element to have attribute(s).</p>
182      */
183     public boolean isNil() {
184         return (value == null) || nil;
185     }
186
187     /**
188      * <p>Set whether this element has nil content.</p>
189      *
190      * @see #isNil()
191      */
192     public void setNil(boolean value) {
193         this.nil = value;
194     }
195
196     /* Convenience methods
197      * (Not necessary but they do unambiguously conceptualize
198      * the rationale behind this class' fields.)
199      */
200
201     /**

```

```

202     * Returns true iff this xml element declaration is global.
203     */
204     public boolean isGlobalScope() {
205         return this.scope == GlobalScope.class;
206     }
207
208     /**
209     * Returns true iff this xml element instance's value has a different
210     * type than xml element declaration's declared type.
211     */
212     public boolean isTypeSubstituted() {
213         if(value==null) return false;
214         return value.getClass() != declaredType;
215     }
216
217     private static final long serialVersionUID = 1L;
218 }

```

## 2.12 JAXBException.java

Secuencia 14: javax/xml/bind/JAXBException.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import java.io.PrintWriter;
29
30 /**
31 * This is the root exception class for all JAXB exceptions.
32 *

```

```
33  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
34  * @see JAXBContext
35  * @see Marshaller
36  * @see Unmarshaller
37  * @since JAXB1.0
38  */
39  public class JAXBException extends Exception {
40
41      /**
42       * Vendor specific error code
43       *
44       */
45      private String errorCode;
46
47      /**
48       * Exception reference
49       *
50       */
51      private volatile Throwable linkedException;
52
53      static final long serialVersionUID = -5621384651494307979L;
54
55      /**
56       * Construct a JAXBException with the specified detail message. The
57       * errorCode and linkedException will default to null.
58       *
59       * @param message a description of the exception
60       */
61      public JAXBException(String message) {
62          this( message, null, null );
63      }
64
65      /**
66       * Construct a JAXBException with the specified detail message and vendor
67       * specific errorCode. The linkedException will default to null.
68       *
69       * @param message a description of the exception
70       * @param errorCode a string specifying the vendor specific error code
71       */
72      public JAXBException(String message, String errorCode) {
73          this( message, errorCode, null );
74      }
75
76      /**
77       * Construct a JAXBException with a linkedException. The detail message and
78       * vendor specific errorCode will default to null.
79       *
80       * @param exception the linked exception
81       */
82      public JAXBException(Throwable exception) {
83          this( null, null, exception );
84      }
85
```

```
86  /**
87   * Construct a JAXBException with the specified detail message and
88   * linkedException. The errorCode will default to null.
89   *
90   * @param message a description of the exception
91   * @param exception the linked exception
92   */
93  public JAXBException(String message, Throwable exception) {
94      this( message, null, exception );
95  }
96
97  /**
98   * Construct a JAXBException with the specified detail message, vendor
99   * specific errorCode, and linkedException.
100  *
101  * @param message a description of the exception
102  * @param errorCode a string specifying the vendor specific error code
103  * @param exception the linked exception
104  */
105  public JAXBException(String message, String errorCode, Throwable exception) {
106      super( message );
107      this.errorCode = errorCode;
108      this.linkedException = exception;
109  }
110
111  /**
112   * Get the vendor specific error code
113   *
114   * @return a string specifying the vendor specific error code
115   */
116  public String getErrorCode() {
117      return this.errorCode;
118  }
119
120  /**
121   * Get the linked exception
122   *
123   * @return the linked Exception, null if none exists
124   */
125  public Throwable getLinkedException() {
126      return linkedException;
127  }
128
129  /**
130   * Add a linked Exception.
131   *
132   * @param exception the linked Exception (A null value is permitted and
133   * indicates that the linked exception does not exist or
134   * is unknown).
135   */
136  public void setLinkedException( Throwable exception ) {
137      this.linkedException = exception;
138  }
```

```

139
140 /**
141  * Returns a short description of this JAXBException.
142  *
143  */
144 public String toString() {
145     return linkedException == null ?
146         super.toString() :
147         super.toString() + "\n - with linked exception:\n[" +
148             linkedException.toString()+ "]";
149 }
150
151 /**
152  * Prints this JAXBException and its stack trace (including the stack trace
153  * of the linkedException if it is non-null) to the PrintStream.
154  *
155  * @param s PrintStream to use for output
156  */
157 public void printStackTrace( java.io.PrintStream s ) {
158     super.printStackTrace(s);
159 }
160
161 /**
162  * Prints this JAXBException and its stack trace (including the stack trace
163  * of the linkedException if it is non-null) to <tt>System.err</tt>.
164  *
165  */
166 public void printStackTrace() {
167     super.printStackTrace();
168 }
169
170 /**
171  * Prints this JAXBException and its stack trace (including the stack trace
172  * of the linkedException if it is non-null) to the PrintWriter.
173  *
174  * @param s PrintWriter to use for output
175  */
176 public void printStackTrace(PrintWriter s) {
177     super.printStackTrace(s);
178 }
179
180 @Override
181 public Throwable getCause() {
182     return linkedException;
183 }
184 }

```

## 2.13 JAXBIntrospector.java

Secuencia 15: javax/xml/bind/JAXBIntrospector.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

```

```

4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27 import javax.xml.namespace.QName;
28
29 /**
30  * Provide access to JAXB xml binding data for a JAXB object.
31  *
32  * <p>
33  * Initially, the intent of this class is to just conceptualize how
34  * a JAXB application developer can access xml binding information,
35  * independent if binding model is java to schema or schema to java.
36  * Since accessing the XML element name related to a JAXB element is
37  * a highly requested feature, demonstrate access to this
38  * binding information.
39  *
40  * The factory method to get a <code>JAXBIntrospector</code> instance is
41  * {@link JAXBContext#createJAXBIntrospector()}.
42  *
43  * @see JAXBContext#createJAXBIntrospector()
44  * @since JAXB2.0
45  */
46 public abstract class JAXBIntrospector {
47
48     /**
49      * <p>Return true if <code>object</code> represents a JAXB element.</p>
50      * <p>Parameter <code>object</code> is a JAXB element for following cases:
51      * <ol>
52      * <li>It is an instance of <code>javax.xml.bind.JAXBElement</code>.</li>
53      * <li>The class of <code>object</code> is annotated with
54      * <code>XmlElement</code>.
55      * </li>
56      * </ol>

```



```

57      *
58      * @see #getElementName(Object)
59      */
60      public abstract boolean isElement(Object object);
61
62      /**
63       * <p>Get xml element qname for <code>jaxbElement</code>.</p>
64       *
65       * @param jaxbElement is an object that {@link #isElement(Object)} returned true.
66       *
67       * @return xml element qname associated with jaxbElement;
68       *         null if <code>jaxbElement</code> is not a JAXB Element.
69       */
70      public abstract QName getElementName(Object jaxbElement);
71
72      /**
73       * <p>Get the element value of a JAXB element.</p>
74       *
75       * <p>Convenience method to abstract whether working with either
76       * a javax.xml.bind.JAXBElement instance or an instance of
77       * <tt>#64XmlRootElement</tt> annotated Java class.</p>
78       *
79       * @param jaxbElement object that #isElement(Object) returns true.
80       *
81       * @return The element value of the <code>jaxbElement</code>.
82       */
83      public static Object getValue(Object jaxbElement) {
84          if (jaxbElement instanceof JAXBElement) {
85              return ((JAXBElement)jaxbElement).getValue();
86          } else {
87              // assume that class of this instance is
88              // annotated with @XmlRootElement.
89              return jaxbElement;
90          }
91      }
92  }

```

## 2.14 JAXBPermission.java

Secuencia 16: javax/xml/bind/JAXBPermission.java

```

1  /*
2  * Copyright (c) 2007, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```

14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind;
27
28 import java.security.BasicPermission;
29
30 /**
31  * This class is for JAXB permissions. A {@code JAXBPermission}
32  * contains a name (also referred to as a "target name") but
33  * no actions list; you either have the named permission
34  * or you don't.
35  *
36  * <P>
37  * The target name is the name of the JAXB permission (see below).
38  *
39  * <P>
40  * The following table lists all the possible {@code JAXBPermission} target names,
41  * and for each provides a description of what the permission allows
42  * and a discussion of the risks of granting code the permission.
43  * <P>
44  *
45  * <table border=1 cellpadding=5 summary="Permission target name, what the permission allows, and
46  *   associated risks">
47  * <tr>
48  * <th>Permission Target Name</th>
49  * <th>What the Permission Allows</th>
50  * <th>Risks of Allowing this Permission</th>
51  * </tr>
52  *
53  * <tr>
54  * <td>setDatatypeConverter</td>
55  * <td>
56  *   Allows the code to set VM-wide {@link DatatypeConverterInterface}
57  *   via {@link DatatypeConverter#setDatatypeConverter(DatatypeConverterInterface)} the
58  *   setDatatypeConverter method}
59  *   that all the methods on {@link DatatypeConverter} uses.
60  * </td>
61  * <td>
62  *   Malicious code can set {@link DatatypeConverterInterface}, which has
63  *   VM-wide singleton semantics, before a genuine JAXB implementation sets one.
64  *   This allows malicious code to gain access to objects that it may otherwise
65  *   not have access to, such as {@link java.awt.Frame#getFrames()} that belongs to
66  *   another application running in the same JVM.

```

```

65 * </td>
66 * </tr>
67 * </table>
68 *
69 * @see java.security.BasicPermission
70 * @see java.security.Permission
71 * @see java.security.Permissions
72 * @see java.security.PermissionCollection
73 * @see java.lang.SecurityManager
74 *
75 * @author Joe Fialli
76 * @since JAXB 2.2
77 */
78
79 /* code was borrowed originally from java.lang.RuntimePermission. */
80 public final class JAXBPermission extends BasicPermission {
81     /**
82      * Creates a new JAXBPermission with the specified name.
83      *
84      * @param name
85      * The name of the JAXBPermission. As of 2.2 only "setDatatypeConverter"
86      * is defined.
87      */
88     public JAXBPermission(String name) {
89         super(name);
90     }
91
92     private static final long serialVersionUID = 1L;
93 }

```

## 2.15 MarshalException.java

Secuencia 17: javax/xml/bind/MarshalException.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```

```

21  *
22  *
23  *
24  */
25
26  package javax.xml.bind;
27
28  /**
29   * This exception indicates that an error has occurred while performing
30   * a marshal operation that the provider is unable to recover from.
31   *
32   * <p>
33   * The <tt>ValidationEventHandler</tt> can cause this exception to be thrown
34   * during the marshal operations. See
35   * {@link ValidationEventHandler#handleEvent(ValidationEvent)}
36   * ValidationEventHandler.handleEvent(ValidationEvent)}.
37   *
38   * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
39   * @see JAXBException
40   * @see Marshaller
41   * @since JAXB1.0
42   */
43  public class MarshalException extends JAXBException {
44
45      /**
46       * Construct a MarshalException with the specified detail message. The
47       * errorCode and linkedException will default to null.
48       *
49       * @param message a description of the exception
50       */
51      public MarshalException( String message ) {
52          this( message, null, null );
53      }
54
55      /**
56       * Construct a MarshalException with the specified detail message and vendor
57       * specific errorCode. The linkedException will default to null.
58       *
59       * @param message a description of the exception
60       * @param errorCode a string specifying the vendor specific error code
61       */
62      public MarshalException( String message, String errorCode ) {
63          this( message, errorCode, null );
64      }
65
66      /**
67       * Construct a MarshalException with a linkedException. The detail message and
68       * vendor specific errorCode will default to null.
69       *
70       * @param exception the linked exception
71       */
72      public MarshalException( Throwable exception ) {
73          this( null, null, exception );

```

```

74     }
75
76     /**
77      * Construct a MarshalException with the specified detail message and
78      * linkedException. The errorCode will default to null.
79      *
80      * @param message a description of the exception
81      * @param exception the linked exception
82      */
83     public MarshalException( String message, Throwable exception ) {
84         this( message, null, exception );
85     }
86
87     /**
88      * Construct a MarshalException with the specified detail message, vendor
89      * specific errorCode, and linkedException.
90      *
91      * @param message a description of the exception
92      * @param errorCode a string specifying the vendor specific error code
93      * @param exception the linked exception
94      */
95     public MarshalException( String message, String errorCode, Throwable exception ) {
96         super( message, errorCode, exception );
97     }
98
99 }

```

## 2.16 Marshaller.java

Secuencia 18: javax/xml/bind/Marshaller.java

```

1  /*
2   * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *

```

```
24  */
25
26  package javax.xml.bind;
27
28  import javax.xml.bind.annotation.XmlRootElement;
29  import javax.xml.bind.annotation.adapters.XmlAdapter;
30  import javax.xml.bind.attachment.AttachmentMarshaller;
31  import javax.xml.validation.Schema;
32  import java.io.File;
33
34  /**
35   * <p>
36   * The <tt>Marshaller</tt> class is responsible for governing the process
37   * of serializing Java content trees back into XML data. It provides the basic
38   * marshalling methods:
39   *
40   * <p>
41   * <i>Assume the following setup code for all following code fragments:</i>
42   * <blockquote>
43   *     <pre>
44   *         JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
45   *         Unmarshaller u = jc.createUnmarshaller();
46   *         Object element = u.unmarshal( new File( "foo.xml" ) );
47   *         Marshaller m = jc.createMarshaller();
48   *     </pre>
49   * </blockquote>
50   *
51   * <p>
52   * Marshalling to a File:
53   * <blockquote>
54   *     <pre>
55   *         OutputStream os = new FileOutputStream( "nosferatu.xml" );
56   *         m.marshal( element, os );
57   *     </pre>
58   * </blockquote>
59   *
60   * <p>
61   * Marshalling to a SAX ContentHandler:
62   * <blockquote>
63   *     <pre>
64   *         // assume MyContentHandler instanceof ContentHandler
65   *         m.marshal( element, new MyContentHandler() );
66   *     </pre>
67   * </blockquote>
68   *
69   * <p>
70   * Marshalling to a DOM Node:
71   * <blockquote>
72   *     <pre>
73   *         DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
74   *         dbf.setNamespaceAware(true);
75   *         DocumentBuilder db = dbf.newDocumentBuilder();
76   *         Document doc = db.newDocument();
```

```
77 *
78 *     m.marshal( element, doc );
79 * </pre>
80 * </blockquote>
81 *
82 * <p>
83 * Marshalling to a java.io.OutputStream:
84 * <blockquote>
85 *     <pre>
86 *         m.marshal( element, System.out );
87 *     </pre>
88 * </blockquote>
89 *
90 * <p>
91 * Marshalling to a java.io.Writer:
92 * <blockquote>
93 *     <pre>
94 *         m.marshal( element, new PrintWriter( System.out ) );
95 *     </pre>
96 * </blockquote>
97 *
98 * <p>
99 * Marshalling to a javax.xml.transform.SAXResult:
100 * <blockquote>
101 *     <pre>
102 *         // assume MyContentHandler instanceof ContentHandler
103 *         SAXResult result = new SAXResult( new MyContentHandler() );
104 *
105 *         m.marshal( element, result );
106 *     </pre>
107 * </blockquote>
108 *
109 * <p>
110 * Marshalling to a javax.xml.transform.DOMResult:
111 * <blockquote>
112 *     <pre>
113 *         DOMResult result = new DOMResult();
114 *
115 *         m.marshal( element, result );
116 *     </pre>
117 * </blockquote>
118 *
119 * <p>
120 * Marshalling to a javax.xml.transform.StreamResult:
121 * <blockquote>
122 *     <pre>
123 *         StreamResult result = new StreamResult( System.out );
124 *
125 *         m.marshal( element, result );
126 *     </pre>
127 * </blockquote>
128 *
129 * <p>
```

```

130 * Marshalling to a javax.xml.stream.XMLStreamWriter:
131 * <blockquote>
132 *   <pre>
133 *       XMLStreamWriter xmlStreamWriter =
134 *           XMLOutputFactory.newInstance().createXMLStreamWriter( ... );
135 *
136 *       m.marshal( element, xmlStreamWriter );
137 *   </pre>
138 * </blockquote>
139 *
140 * <p>
141 * Marshalling to a javax.xml.stream.XMLEventWriter:
142 * <blockquote>
143 *   <pre>
144 *       XMLEventWriter xmlEventWriter =
145 *           XMLOutputFactory.newInstance().createXMLEventWriter( ... );
146 *
147 *       m.marshal( element, xmlEventWriter );
148 *   </pre>
149 * </blockquote>
150 *
151 * <p>
152 * <a name="elementMarshalling"></a>
153 * <b>Marshalling content tree rooted by a JAXB element</b><br>
154 * <blockquote>
155 * The first parameter of the overloaded
156 * <tt>Marshaller.marshal(java.lang.Object, ...)</tt> methods must be a
157 * JAXB element as computed by
158 * {@link JAXBIntrospector#isElement(java.lang.Object)};
159 * otherwise, a <tt>Marshaller.marshal</tt> method must throw a
160 * {@link MarshalException}. There exist two mechanisms
161 * to enable marshalling an instance that is not a JAXB element.
162 * One method is to wrap the instance as a value of a {@link JAXBElement},
163 * and pass the wrapper element as the first parameter to
164 * a <tt>Marshaller.marshal</tt> method. For java to schema binding, it
165 * is also possible to simply annotate the instance's class with
166 * &#64;{@link XmlRootElement}.
167 * </blockquote>
168 *
169 * <p>
170 * <b>Encoding</b><br>
171 * <blockquote>
172 * By default, the Marshaller will use UTF-8 encoding when generating XML data
173 * to a <tt>java.io.OutputStream</tt>, or a <tt>java.io.Writer</tt>. Use the
174 * {@link #setProperty(String,Object) setProperty} API to change the output
175 * encoding used during these marshal operations. Client applications are
176 * expected to supply a valid character encoding name as defined in the
177 * <a href="http://www.w3.org/TR/2000/REC-xml-20001006#charencoding">W3C XML 1.0
178 * Recommendation</a> and supported by your Java Platform</a>.
179 * </blockquote>
180 *
181 * <p>
182 * <b>Validation and Well-Formedness</b><br>

```



```

183 * <blockquote>
184 * <p>
185 * Client applications are not required to validate the Java content tree prior
186 * to calling any of the marshal API's. Furthermore, there is no requirement
187 * that the Java content tree be valid with respect to its original schema in
188 * order to marshal it back into XML data. Different JAXB Providers will
189 * support marshalling invalid Java content trees at varying levels, however
190 * all JAXB Providers must be able to marshal a valid content tree back to
191 * XML data. A JAXB Provider must throw a <tt>MarshalException</tt> when it
192 * is unable to complete the marshal operation due to invalid content. Some
193 * JAXB Providers will fully allow marshalling invalid content, others will fail
194 * on the first validation error.
195 * <p>
196 * Even when schema validation is not explicitly enabled for the marshal operation,
197 * it is possible that certain types of validation events will be detected
198 * during the operation. Validation events will be reported to the registered
199 * event handler. If the client application has not registered an event handler
200 * prior to invoking one of the marshal API's, then events will be delivered to
201 * a default event handler which will terminate the marshal operation after
202 * encountering the first error or fatal error. Note that for JAXB 2.0 and
203 * later versions, {@link javax.xml.bind.helpers.DefaultValidationEventHandler} is
204 * no longer used.
205 *
206 * </blockquote>
207 *
208 * <p>
209 * <a name="supportedProps"></a>
210 * <b>Supported Properties</b><br>
211 * <blockquote>
212 * <p>
213 * All JAXB Providers are required to support the following set of properties.
214 * Some providers may support additional properties.
215 * <dl>
216 * <dt><tt>jaxb.encoding</tt> - value must be a java.lang.String</dt>
217 * <dd>The output encoding to use when marshalling the XML data. The
218 * Marshaller will use "UTF-8" by default if this property is not
219 * specified.</dd>
220 * <dt><tt>jaxb.formatted.output</tt> - value must be a java.lang.Boolean</dt>
221 * <dd>This property controls whether or not the Marshaller will format
222 * the resulting XML data with line breaks and indentation. A
223 * true value for this property indicates human readable indented
224 * xml data, while a false value indicates unformatted xml data.
225 * The Marshaller will default to false (unformatted) if this
226 * property is not specified.</dd>
227 * <dt><tt>jaxb.schemaLocation</tt> - value must be a java.lang.String</dt>
228 * <dd>This property allows the client application to specify an
229 * xsi:schemaLocation attribute in the generated XML data. The format of
230 * the schemaLocation attribute value is discussed in an easy to
231 * understand, non-normative form in
232 * <a href="http://www.w3.org/TR/xmlschema-0/#schemaLocation">Section 5.6
233 * of the W3C XML Schema Part 0: Primer</a> and specified in
234 * <a href="http://www.w3.org/TR/xmlschema-1/#Instance_Document_Constructions">
235 * Section 2.6 of the W3C XML Schema Part 1: Structures</a>.</dd>

```

```

236 * <dt><tt>jaxb.noNamespaceSchemaLocation</tt> - value must be a java.lang.String</dt>
237 * <dd>This property allows the client application to specify an
238 *     xsi:noNamespaceSchemaLocation attribute in the generated XML
239 *     data. The format of the schemaLocation attribute value is discussed in
240 *     an easy to understand, non-normative form in
241 *     <a href="http://www.w3.org/TR/xmlschema-0/#schemaLocation">Section 5.6
242 *     of the W3C XML Schema Part 0: Primer</a> and specified in
243 *     <a href="http://www.w3.org/TR/xmlschema-1/#Instance_Document_Constructions">
244 *     Section 2.6 of the W3C XML Schema Part 1: Structures</a>.</dd>
245 * <dt><tt>jaxb.fragment</tt> - value must be a java.lang.Boolean</dt>
246 * <dd>This property determines whether or not document level events will be
247 *     generated by the Marshaller. If the property is not specified, the
248 *     default is <tt>>false</tt>. This property has different implications depending
249 *     on which marshal api you are using - when this property is set to true:<br>
250 *     <ul>
251 *         <li>{@link #marshal(Object,org.xml.sax.ContentHandler) marshal(Object,ContentHandler)} -
252 *             the Marshaller won't
253 *             invoke {@link org.xml.sax.ContentHandler#startDocument()} and
254 *             {@link org.xml.sax.ContentHandler#endDocument()}.</li>
255 *         <li>{@link #marshal(Object,org.w3c.dom.Node) marshal(Object,Node)} - the property has no
256 *             effect on this
257 *             API.</li>
258 *         <li>{@link #marshal(Object,java.io.OutputStream) marshal(Object,OutputStream)} - the
259 *             Marshaller won't
260 *             generate an xml declaration.</li>
261 *         <li>{@link #marshal(Object,java.io.Writer) marshal(Object,Writer)} - the Marshaller won't
262 *             generate an xml declaration.</li>
263 *         <li>{@link #marshal(Object,javax.xml.transform.Result) marshal(Object,Result)} - depends
264 *             on the kind of
265 *             Result object, see semantics for Node, ContentHandler, and Stream APIs</li>
266 *         <li>{@link #marshal(Object,javax.xml.stream.XMLEventWriter) marshal(Object,
267 *             XMLEventWriter)} - the
268 *             Marshaller will not generate {@link javax.xml.stream.events.XMLEvent#START_DOCUMENT}
269 *             and
270 *             {@link javax.xml.stream.events.XMLEvent#END_DOCUMENT} events.</li>
271 *         <li>{@link #marshal(Object,javax.xml.stream.XMLStreamWriter) marshal(Object,
272 *             XMLStreamWriter)} - the
273 *             Marshaller will not generate {@link javax.xml.stream.events.XMLEvent#START_DOCUMENT}
274 *             and
275 *             {@link javax.xml.stream.events.XMLEvent#END_DOCUMENT} events.</li>
276 *     </ul>
277 * </dd>
278 * </dl>
279 * </blockquote>
280 *
281 * <p>
282 * <a name="marshalEventCallback"></a>
283 * <b>Marshal Event Callbacks</b><br>
284 * <blockquote>
285 * "The {@link Marshaller} provides two styles of callback mechanisms
286 * that allow application specific processing during key points in the
287 * unmarshalling process. In 'class defined' event callbacks, application

```

```

280 * specific code placed in JAXB mapped classes is triggered during
281 * marshalling. 'External listeners' allow for centralized processing
282 * of marshal events in one callback method rather than by type event callbacks.
283 *
284 * <p>
285 * Class defined event callback methods allow any JAXB mapped class to specify
286 * its own specific callback methods by defining methods with the following method signatures:
287 * <blockquote>
288 * <pre>
289 * // Invoked by Marshaller after it has created an instance of this object.
290 * boolean beforeMarshal(Marshaller);
291 *
292 * // Invoked by Marshaller after it has marshalled all properties of this object.
293 * void afterMarshal(Marshaller);
294 * </pre>
295 * </blockquote>
296 * The class defined event callback methods should be used when the callback method requires
297 * access to non-public methods and/or fields of the class.
298 * <p>
299 * The external listener callback mechanism enables the registration of a {@link Listener}
300 * instance with a {@link Marshaller#setListener(Listener)}. The external listener receives all
301 * callback events,
302 * allowing for more centralized processing than per class defined callback methods.
303 * <p>
304 * The 'class defined' and external listener event callback methods are independent of each other,
305 * both can be called for one event. The invocation ordering when both listener callback methods
306 * exist is
307 * defined in {@link Listener#beforeMarshal(Object)} and {@link Listener#afterMarshal(Object)}.
308 * <p>
309 * An event callback method throwing an exception terminates the current marshal process.
310 * </blockquote>
311 *
312 * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Ryan Shoemaker, Sun
313 * Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
314 *
315 * @see JAXBContext
316 * @see Validator
317 * @see Unmarshaller
318 * @since JAXB1.0
319 */
320 public interface Marshaller {
321
322     /**
323      * The name of the property used to specify the output encoding in
324      * the marshalled XML data.
325      */
326     public static final String JAXB_ENCODING =
327         "jaxb.encoding";
328
329     /**
330      * The name of the property used to specify whether or not the marshalled
331      * XML data is formatted with linefeeds and indentation.
332      */
333     public static final String JAXB_FORMATTED_OUTPUT =

```

```

330     "jaxb.formatted.output";
331
332     /**
333      * The name of the property used to specify the xsi:schemaLocation
334      * attribute value to place in the marshalled XML output.
335      */
336     public static final String JAXB_SCHEMA_LOCATION =
337         "jaxb.schemaLocation";
338
339     /**
340      * The name of the property used to specify the
341      * xsi:noNamespaceSchemaLocation attribute value to place in the marshalled
342      * XML output.
343      */
344     public static final String JAXB_NO_NAMESPACE_SCHEMA_LOCATION =
345         "jaxb.noNamespaceSchemaLocation";
346
347     /**
348      * The name of the property used to specify whether or not the marshaller
349      * will generate document level events (ie calling startDocument or endDocument).
350      */
351     public static final String JAXB_FRAGMENT =
352         "jaxb.fragment";
353
354     /**
355      * Marshal the content tree rooted at <tt>jaxbElement</tt> into the specified
356      * <tt>javax.xml.transform.Result</tt>.
357      *
358      * <p>
359      * All JAXB Providers must at least support
360      * {@link javax.xml.transform.dom.DOMResult},
361      * {@link javax.xml.transform.sax.SAXResult}, and
362      * {@link javax.xml.transform.stream.StreamResult}. It can
363      * support other derived classes of <tt>Result</tt> as well.
364      *
365      * @param jaxbElement
366      *     The root of content tree to be marshalled.
367      * @param result
368      *     XML will be sent to this Result
369      *
370      * @throws JAXBException
371      *     If any unexpected problem occurs during the marshalling.
372      * @throws MarshalException
373      *     If the {@link ValidationEventHandler ValidationEventHandler}
374      *     returns false from its <tt>handleEvent</tt> method or the
375      *     <tt>Marshaller</tt> is unable to marshal <tt>obj</tt> (or any
376      *     object reachable from <tt>obj</tt>). See <a href="#elementMarshalling">
377      *     Marshalling a JAXB element</a>.
378      * @throws IllegalArgumentException
379      *     If any of the method parameters are null
380      */
381     public void marshal( Object jaxbElement, javax.xml.transform.Result result )
382         throws JAXBException;

```

```

383
384 /**
385  * Marshal the content tree rooted at <tt>jaxbElement</tt> into an output stream.
386  *
387  * @param jaxbElement
388  *     The root of content tree to be marshalled.
389  * @param os
390  *     XML will be added to this stream.
391  *
392  * @throws JAXBException
393  *     If any unexpected problem occurs during the marshalling.
394  * @throws MarshalException
395  *     If the {@link ValidationEventHandler ValidationEventHandler}
396  *     returns false from its <tt>handleEvent</tt> method or the
397  *     <tt>Marshaller</tt> is unable to marshal <tt>obj</tt> (or any
398  *     object reachable from <tt>obj</tt>). See <a href="#elementMarshalling">
399  *     Marshalling a JAXB element</a>.
400  * @throws IllegalArgumentException
401  *     If any of the method parameters are null
402  */
403 public void marshal( Object jaxbElement, java.io.OutputStream os )
404     throws JAXBException;
405
406 /**
407  * Marshal the content tree rooted at <tt>jaxbElement</tt> into a file.
408  *
409  * @param jaxbElement
410  *     The root of content tree to be marshalled.
411  * @param output
412  *     File to be written. If this file already exists, it will be overwritten.
413  *
414  * @throws JAXBException
415  *     If any unexpected problem occurs during the marshalling.
416  * @throws MarshalException
417  *     If the {@link ValidationEventHandler ValidationEventHandler}
418  *     returns false from its <tt>handleEvent</tt> method or the
419  *     <tt>Marshaller</tt> is unable to marshal <tt>obj</tt> (or any
420  *     object reachable from <tt>obj</tt>). See <a href="#elementMarshalling">
421  *     Marshalling a JAXB element</a>.
422  * @throws IllegalArgumentException
423  *     If any of the method parameters are null
424  * @since JAXB2.1
425  */
426 public void marshal( Object jaxbElement, File output )
427     throws JAXBException;
428
429 /**
430  * Marshal the content tree rooted at <tt>jaxbElement</tt> into a Writer.
431  *
432  * @param jaxbElement
433  *     The root of content tree to be marshalled.
434  * @param writer
435  *     XML will be sent to this writer.

```

```

436 *
437 * @throws JAXBException
438 *     If any unexpected problem occurs during the marshalling.
439 * @throws MarshalException
440 *     If the {@link ValidationEventHandler ValidationEventHandler}
441 *     returns false from its <tt>handleEvent</tt> method or the
442 *     <tt>Marshaller</tt> is unable to marshal <tt>obj</tt> (or any
443 *     object reachable from <tt>obj</tt>). See <a href="#elementMarshalling">
444 *     Marshalling a JAXB element</a>.
445 * @throws IllegalArgumentException
446 *     If any of the method parameters are null
447 */
448 public void marshal( Object jaxbElement, java.io.Writer writer )
449     throws JAXBException;
450
451 /**
452 * Marshal the content tree rooted at <tt>jaxbElement</tt> into SAX2 events.
453 *
454 * @param jaxbElement
455 *     The root of content tree to be marshalled.
456 * @param handler
457 *     XML will be sent to this handler as SAX2 events.
458 *
459 * @throws JAXBException
460 *     If any unexpected problem occurs during the marshalling.
461 * @throws MarshalException
462 *     If the {@link ValidationEventHandler ValidationEventHandler}
463 *     returns false from its <tt>handleEvent</tt> method or the
464 *     <tt>Marshaller</tt> is unable to marshal <tt>obj</tt> (or any
465 *     object reachable from <tt>obj</tt>). See <a href="#elementMarshalling">
466 *     Marshalling a JAXB element</a>.
467 * @throws IllegalArgumentException
468 *     If any of the method parameters are null
469 */
470 public void marshal( Object jaxbElement, org.xml.sax.ContentHandler handler )
471     throws JAXBException;
472
473 /**
474 * Marshal the content tree rooted at <tt>jaxbElement</tt> into a DOM tree.
475 *
476 * @param jaxbElement
477 *     The content tree to be marshalled.
478 * @param node
479 *     DOM nodes will be added as children of this node.
480 *     This parameter must be a Node that accepts children
481 *     ({@link org.w3c.dom.Document},
482 *     {@link org.w3c.dom.DocumentFragment}, or
483 *     {@link org.w3c.dom.Element})
484 *
485 * @throws JAXBException
486 *     If any unexpected problem occurs during the marshalling.
487 * @throws MarshalException
488 *     If the {@link ValidationEventHandler ValidationEventHandler}

```

```

489     *     returns false from its <tt>handleEvent</tt> method or the
490     *     <tt>Marshaller</tt> is unable to marshal <tt>jaxbElement</tt> (or any
491     *     object reachable from <tt>jaxbElement</tt>). See <a href="#elementMarshalling">
492     *     Marshalling a JAXB element</a>.
493     * @throws IllegalArgumentException
494     *     If any of the method parameters are null
495     */
496     public void marshal( Object jaxbElement, org.w3c.dom.Node node )
497         throws JAXBException;
498
499     /**
500     * Marshal the content tree rooted at <tt>jaxbElement</tt> into a
501     * {@link javax.xml.stream.XMLStreamWriter}.
502     *
503     * @param jaxbElement
504     *     The content tree to be marshalled.
505     * @param writer
506     *     XML will be sent to this writer.
507     *
508     * @throws JAXBException
509     *     If any unexpected problem occurs during the marshalling.
510     * @throws MarshalException
511     *     If the {@link ValidationEventHandler ValidationEventHandler}
512     *     returns false from its <tt>handleEvent</tt> method or the
513     *     <tt>Marshaller</tt> is unable to marshal <tt>obj</tt> (or any
514     *     object reachable from <tt>obj</tt>). See <a href="#elementMarshalling">
515     *     Marshalling a JAXB element</a>.
516     * @throws IllegalArgumentException
517     *     If any of the method parameters are null
518     * @since JAXB 2.0
519     */
520     public void marshal( Object jaxbElement, javax.xml.stream.XMLStreamWriter writer )
521         throws JAXBException;
522
523     /**
524     * Marshal the content tree rooted at <tt>jaxbElement</tt> into a
525     * {@link javax.xml.stream.XMLEventWriter}.
526     *
527     * @param jaxbElement
528     *     The content tree rooted at jaxbElement to be marshalled.
529     * @param writer
530     *     XML will be sent to this writer.
531     *
532     * @throws JAXBException
533     *     If any unexpected problem occurs during the marshalling.
534     * @throws MarshalException
535     *     If the {@link ValidationEventHandler ValidationEventHandler}
536     *     returns false from its <tt>handleEvent</tt> method or the
537     *     <tt>Marshaller</tt> is unable to marshal <tt>obj</tt> (or any
538     *     object reachable from <tt>obj</tt>). See <a href="#elementMarshalling">
539     *     Marshalling a JAXB element</a>.
540     * @throws IllegalArgumentException
541     *     If any of the method parameters are null

```



```
542     * @since JAXB 2.0
543     */
544     public void marshal( Object jaxbElement, javax.xml.stream.XMLEventWriter writer )
545         throws JAXBException;
546
547     /**
548     * Get a DOM tree view of the content tree(Optional).
549     *
550     * If the returned DOM tree is updated, these changes are also
551     * visible in the content tree.
552     * Use {@link #marshal(Object, org.w3c.dom.Node)} to force
553     * a deep copy of the content tree to a DOM representation.
554     *
555     * @param contentTree - JAXB Java representation of XML content
556     *
557     * @return the DOM tree view of the contentTree
558     *
559     * @throws UnsupportedOperationException
560     *         If the JAXB provider implementation does not support a
561     *         DOM view of the content tree
562     *
563     * @throws IllegalArgumentException
564     *         If any of the method parameters are null
565     *
566     * @throws JAXBException
567     *         If any unexpected problem occurs
568     */
569
570     public org.w3c.dom.Node getNode( java.lang.Object contentTree )
571         throws JAXBException;
572
573     /**
574     * Set the particular property in the underlying implementation of
575     * <tt>Marshaller</tt>. This method can only be used to set one of
576     * the standard JAXB defined properties above or a provider specific
577     * property. Attempting to set an undefined property will result in
578     * a PropertyException being thrown. See <a href="#supportedProps">
579     * Supported Properties</a>.
580     *
581     * @param name the name of the property to be set. This value can either
582     *         be specified using one of the constant fields or a user
583     *         supplied string.
584     * @param value the value of the property to be set
585     *
586     * @throws PropertyException when there is an error processing the given
587     *         property or value
588     * @throws IllegalArgumentException
589     *         If the name parameter is null
590     */
591     public void setProperty( String name, Object value )
592         throws PropertyException;
593
594     /**
```



```

595 * Get the particular property in the underlying implementation of
596 * <tt>Marshaller</tt>. This method can only be used to get one of
597 * the standard JAXB defined properties above or a provider specific
598 * property. Attempting to get an undefined property will result in
599 * a PropertyException being thrown. See <a href="#supportedProps">
600 * Supported Properties</a>.
601 *
602 * @param name the name of the property to retrieve
603 * @return the value of the requested property
604 *
605 * @throws PropertyException
606 *         when there is an error retrieving the given property or value
607 *         property name
608 * @throws IllegalArgumentException
609 *         If the name parameter is null
610 */
611 public Object getProperty( String name ) throws PropertyException;
612
613 /**
614 * Allow an application to register a validation event handler.
615 * <p>
616 * The validation event handler will be called by the JAXB Provider if any
617 * validation errors are encountered during calls to any of the marshal
618 * API's. If the client application does not register a validation event
619 * handler before invoking one of the marshal methods, then validation
620 * events will be handled by the default event handler which will terminate
621 * the marshal operation after the first error or fatal error is encountered.
622 * <p>
623 * Calling this method with a null parameter will cause the Marshaller
624 * to revert back to the default default event handler.
625 *
626 * @param handler the validation event handler
627 * @throws JAXBException if an error was encountered while setting the
628 *         event handler
629 */
630 public void setEventHandler( ValidationEventHandler handler )
631     throws JAXBException;
632
633 /**
634 * Return the current event handler or the default event handler if one
635 * hasn't been set.
636 *
637 * @return the current ValidationEventHandler or the default event handler
638 *         if it hasn't been set
639 * @throws JAXBException if an error was encountered while getting the
640 *         current event handler
641 */
642 public ValidationEventHandler getEventHandler()
643     throws JAXBException;
644
645
646
647 /**

```

```

648     * Associates a configured instance of {@link XmlAdapter} with this marshaller.
649     *
650     * <p>
651     * This is a convenience method that invokes <code>setAdapter(adapter.getClass(),adapter);</code>.
652     *
653     * @see #setAdapter(Class,XmlAdapter)
654     * @throws IllegalArgumentException
655     *         if the adapter parameter is null.
656     * @throws UnsupportedOperationException
657     *         if invoked against a JAXB 1.0 implementation.
658     * @since JAXB 2.0
659     */
660     public void setAdapter( XmlAdapter adapter );
661
662     /**
663     * Associates a configured instance of {@link XmlAdapter} with this marshaller.
664     *
665     * <p>
666     * Every marshaller internally maintains a
667     * {@link java.util.Map}&lt;{@link Class},{@link XmlAdapter}>,
668     * which it uses for marshalling classes whose fields/methods are annotated
669     * with {@link javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter}.
670     *
671     * <p>
672     * This method allows applications to use a configured instance of {@link XmlAdapter}.
673     * When an instance of an adapter is not given, a marshaller will create
674     * one by invoking its default constructor.
675     *
676     * @param type
677     *        The type of the adapter. The specified instance will be used when
678     *        {@link javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter#value()}
679     *        refers to this type.
680     * @param adapter
681     *        The instance of the adapter to be used. If null, it will un-register
682     *        the current adapter set for this type.
683     * @throws IllegalArgumentException
684     *         if the type parameter is null.
685     * @throws UnsupportedOperationException
686     *         if invoked against a JAXB 1.0 implementation.
687     * @since JAXB 2.0
688     */
689     public <A extends XmlAdapter> void setAdapter( Class<A> type, A adapter );
690
691     /**
692     * Gets the adapter associated with the specified type.
693     *
694     * This is the reverse operation of the {@link #setAdapter} method.
695     *
696     * @throws IllegalArgumentException
697     *         if the type parameter is null.
698     * @throws UnsupportedOperationException
699     *         if invoked against a JAXB 1.0 implementation.

```

```

700     * @since JAXB 2.0
701     */
702     public <A extends XmlAdapter> A getAdapter( Class<A> type );
703
704
705     /**
706     * <p>Associate a context that enables binary data within an XML document
707     * to be transmitted as XML-binary optimized attachment.
708     * The attachment is referenced from the XML document content model
709     * by content-id URIs(cid) references stored within the xml document.
710     *
711     * @throws IllegalStateException if attempt to concurrently call this
712     *         method during a marshal operation.
713     */
714     void setAttachmentMarshaller(AttachmentMarshaller am);
715
716     AttachmentMarshaller getAttachmentMarshaller();
717
718     /**
719     * Specify the JAXP 1.3 {@link javax.xml.validation.Schema Schema}
720     * object that should be used to validate subsequent marshal operations
721     * against. Passing null into this method will disable validation.
722     *
723     * <p>
724     * This method allows the caller to validate the marshalled XML as it's marshalled.
725     *
726     * <p>
727     * Initially this property is set to <tt>null</tt>.
728     *
729     * @param schema Schema object to validate marshal operations against or null to disable
730     *         validation
731     * @throws UnsupportedOperationException could be thrown if this method is
732     *         invoked on an Marshaller created from a JAXBContext referencing
733     *         JAXB 1.0 mapped classes
734     * @since JAXB2.0
735     */
736     public void setSchema( Schema schema );
737
738     /**
739     * Get the JAXP 1.3 {@link javax.xml.validation.Schema Schema} object
740     * being used to perform marshal-time validation. If there is no
741     * Schema set on the marshaller, then this method will return null
742     * indicating that marshal-time validation will not be performed.
743     *
744     * @return the Schema object being used to perform marshal-time
745     *         validation or null if not present.
746     * @throws UnsupportedOperationException could be thrown if this method is
747     *         invoked on an Marshaller created from a JAXBContext referencing
748     *         JAXB 1.0 mapped classes
749     * @since JAXB2.0
750     */
751     public Schema getSchema();

```

```

752  /**
753   * <p/>
754   * Register an instance of an implementation of this class with a {@link Marshaller} to
       externally listen
755   * for marshal events.
756   * <p/>
757   * <p/>
758   * This class enables pre and post processing of each marshalled object.
759   * The event callbacks are called when marshalling from an instance that maps to an xml element
       or
760   * complex type definition. The event callbacks are not called when marshalling from an
       instance of a
761   * Java datatype that represents a simple type definition.
762   * <p/>
763   * <p/>
764   * External listener is one of two different mechanisms for defining marshal event callbacks.
765   * See <a href="Marshaller.html#marshalEventCallback">Marshal Event Callbacks</a> for an
       overview.
766   *
767   * @see Marshaller#setListener(Listener)
768   * @see Marshaller#getListener()
769   * @since JAXB2.0
770   */
771  public static abstract class Listener {
772      /**
773       * <p/>
774       * Callback method invoked before marshalling from <tt>source</tt> to XML.
775       * <p/>
776       * <p/>
777       * This method is invoked just before marshalling process starts to marshal <tt>source</tt>
       >.
778       * Note that if the class of <tt>source</tt> defines its own <tt>beforeMarshal</tt> method,
779       * the class specific callback method is invoked just before this method is invoked.
780       *
781       * @param source instance of JAXB mapped class prior to marshalling from it.
782       */
783      public void beforeMarshal(Object source) {
784      }
785
786      /**
787       * <p/>
788       * Callback method invoked after marshalling <tt>source</tt> to XML.
789       * <p/>
790       * <p/>
791       * This method is invoked after <tt>source</tt> and all its descendants have been
       marshalled.
792       * Note that if the class of <tt>source</tt> defines its own <tt>afterMarshal</tt> method,
793       * the class specific callback method is invoked just before this method is invoked.
794       *
795       * @param source instance of JAXB mapped class after marshalling it.
796       */
797      public void afterMarshal(Object source) {
798      }

```

```

799     }
800
801     /**
802     * <p>
803     * Register marshal event callback {@link Listener} with this {@link Marshaller}.
804     *
805     * <p>
806     * There is only one Listener per Marshaller. Setting a Listener replaces the previous set
807     * Listener.
808     * One can unregister current Listener by setting listener to <tt>null</tt>.
809     *
810     * @param listener an instance of a class that implements {@link Listener}
811     * @since JAXB2.0
812     */
813     public void setListener(Listener listener);
814
815     /**
816     * <p>Return {@link Listener} registered with this {@link Marshaller}.
817     *
818     * @return registered {@link Listener} or <code>null</code> if no Listener is registered with
819     * this Marshaller.
820     * @since JAXB2.0
821     */
822     public Listener getListener();
823 }

```

## 2.17 Messages.java

Secuencia 19: javax/xml/bind/Messages.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */

```

```
25
26 package javax.xml.bind;
27
28 import java.text.MessageFormat;
29 import java.util.ResourceBundle;
30
31 /**
32  * Formats error messages.
33  */
34 class Messages
35 {
36     static String format( String property ) {
37         return format( property, null );
38     }
39
40     static String format( String property, Object arg1 ) {
41         return format( property, new Object[]{arg1} );
42     }
43
44     static String format( String property, Object arg1, Object arg2 ) {
45         return format( property, new Object[]{arg1,arg2} );
46     }
47
48     static String format( String property, Object arg1, Object arg2, Object arg3 ) {
49         return format( property, new Object[]{arg1,arg2,arg3} );
50     }
51
52     // add more if necessary.
53
54     /** Loads a string resource and formats it with specified arguments. */
55     static String format( String property, Object[] args ) {
56         String text = ResourceBundle.getBundle(Messages.class.getName()).getString(property);
57         return MessageFormat.format(text,args);
58     }
59
60     //
61     //
62     // Message resources
63     //
64     //
65     static final String PROVIDER_NOT_FOUND = // 1 arg
66         "ContextFinder.ProviderNotFound";
67
68     static final String COULD_NOT_INSTANTIATE = // 2 args
69         "ContextFinder.CouldNotInstantiate";
70
71     static final String CANT_FIND_PROPERTIES_FILE = // 1 arg
72         "ContextFinder.CantFindPropertiesFile";
73
74     static final String CANT_MIX_PROVIDERS = // 0 args
75         "ContextFinder.CantMixProviders";
76
77     static final String MISSING_PROPERTY = // 2 args
```

```

78     "ContextFinder.MissingProperty";
79
80     static final String NO_PACKAGE_IN_CONTEXTPATH = // 0 args
81         "ContextFinder.NoPackageInContextPath";
82
83     static final String NAME_VALUE = // 2 args
84         "PropertyException.NameValue";
85
86     static final String CONVERTER_MUST_NOT_BE_NULL = // 0 args
87         "DatatypeConverter.ConverterMustBeNull";
88
89     static final String ILLEGAL_CAST = // 2 args
90         "JAXBContext.IllegalCast";
91 }

```

## 2.18 NotIdentifiableEvent.java

Secuencia 20: javax/xml/bind/NotIdentifiableEvent.java

```

1  /*
2   * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind;
27
28 /**
29  * This event indicates that a problem was encountered resolving an ID/IDREF.
30  *
31  *
32  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
33  *     Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
34  * @see Validator
35  * @see ValidationEventHandler

```

```

35  * @since JAXB1.0
36  */
37  public interface NotIdentifiableEvent extends ValidationEvent {
38
39  }

```

## 2.19 ParseConversionEvent.java

Secuencia 21: javax/xml/bind/ParseConversionEvent.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 /**
29  * This event indicates that a problem was encountered while converting a
30  * string from the XML data into a value of the target Java data type.
31  *
32  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
33  * Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
34  * @see ValidationEvent
35  * @see ValidationEventHandler
36  * @see Unmarshaller
37  * @since JAXB1.0
38  */
39  public interface ParseConversionEvent extends ValidationEvent {
40

```

## 2.20 PrintConversionEvent.java



## Secuencia 22: javax/xml/bind/PrintConversionEvent.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 /**
29 * This event indicates that a problem was encountered while converting data
30 * from the Java content tree into its lexical representation.
31 *
32 * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
33 *     Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
34 * @see ValidationEvent
35 * @see ValidationEventHandler
36 * @see Marshaller
37 * @since JAXB1.0
38 */
39 public interface PrintConversionEvent extends ValidationEvent {
40 }

```

## 2.21 PropertyException.java

## Secuencia 23: javax/xml/bind/PropertyException.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *

```

```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28
29
30 /**
31  * This exception indicates that an error was encountered while getting or
32  * setting a property.
33  *
34  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
35  * Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
36  * @see JAXBContext
37  * @see Validator
38  * @see Unmarshaller
39  * @since JAXB1.0
40  */
41 public class PropertyException extends JAXBException {
42     /**
43      * Construct a PropertyException with the specified detail message. The
44      * errorCode and linkedException will default to null.
45      *
46      * @param message a description of the exception
47      */
48     public PropertyException(String message) {
49         super(message);
50     }
51
52     /**
53      * Construct a PropertyException with the specified detail message and
54      * vendor specific errorCode. The linkedException will default to null.
55      *
56      * @param message a description of the exception
57      * @param errorCode a string specifying the vendor specific error code
58      */
59     public PropertyException(String message, String errorCode) {
```

```
60         super(message, errorCode);
61     }
62
63     /**
64      * Construct a PropertyException with a linkedException. The detail
65      * message and vendor specific errorCode will default to null.
66      *
67      * @param exception the linked exception
68      */
69     public PropertyException(Throwable exception) {
70         super(exception);
71     }
72
73     /**
74      * Construct a PropertyException with the specified detail message and
75      * linkedException. The errorCode will default to null.
76      *
77      * @param message a description of the exception
78      * @param exception the linked exception
79      */
80     public PropertyException(String message, Throwable exception) {
81         super(message, exception);
82     }
83
84     /**
85      * Construct a PropertyException with the specified detail message, vendor
86      * specific errorCode, and linkedException.
87      *
88      * @param message a description of the exception
89      * @param errorCode a string specifying the vendor specific error code
90      * @param exception the linked exception
91      */
92     public PropertyException(
93         String message,
94         String errorCode,
95         Throwable exception) {
96         super(message, errorCode, exception);
97     }
98
99     /**
100      * Construct a PropertyException whose message field is set based on the
101      * name of the property and value.toString().
102      *
103      * @param name the name of the property related to this exception
104      * @param value the value of the property related to this exception
105      */
106     public PropertyException(String name, Object value) {
107         super( Messages.format( Messages.NAME_VALUE,
108                                 name,
109                                 value.toString() ) );
110     }
111
112
```

113 }

## 2.22 SchemaOutputResolver.java

Secuencia 24: javax/xml/bind/SchemaOutputResolver.java

```
1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import javax.xml.transform.Result;
29 import java.io.IOException;
30
31 /**
32 * Controls where a JAXB implementation puts the generated
33 * schema files.
34 *
35 * <p>
36 * An implementation of this abstract class has to be provided by the calling
37 * application to generate schemas.
38 *
39 * <p>
40 * This is a class, not an interface so as to allow future versions to evolve
41 * without breaking the compatibility.
42 *
43 * @author
44 *     Kohsuke Kawaguchi (kohsuke.kawaguchi@sun.com)
45 */
46 public abstract class SchemaOutputResolver {
47     /**
48      * Decides where the schema file (of the given namespace URI)
```

```

49     * will be written, and return it as a {@link Result} object.
50     *
51     * <p>
52     * This method is called only once for any given namespace.
53     * IOW, all the components in one namespace is always written
54     * into the same schema document.
55     *
56     * @param namespaceUri
57     *     The namespace URI that the schema declares.
58     *     Can be the empty string, but never be null.
59     * @param suggestedFileName
60     *     A JAXB implementation generates an unique file name (like "schema1.xsd")
61     *     for the convenience of the callee. This name can be
62     *     used for the file name of the schema, or the callee can just
63     *     ignore this name and come up with its own name.
64     *     This is just a hint.
65     *
66     * @return
67     *     a {@link Result} object that encapsulates the actual destination
68     *     of the schema.
69     *
70     *     If the {@link Result} object has a system ID, it must be an
71     *     absolute system ID. Those system IDs are relativized by the caller and used
72     *     for <xs:import> statements.
73     *
74     *     If the {@link Result} object does not have a system ID, a schema
75     *     for the namespace URI is generated but it won't be explicitly
76     *     <xs:import>ed from other schemas.
77     *
78     *     If {@code null} is returned, the schema generation for this
79     *     namespace URI will be skipped.
80     */
81     public abstract Result createOutput( String namespaceUri, String suggestedFileName ) throws
        IOException;
82 }

```

## 2.23 TypeConstraintException.java

Secuencia 25: javax/xml/bind/TypeConstraintException.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```

```
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind;
27
28 /**
29  * This exception indicates that a violation of a dynamically checked type
30  * constraint was detected.
31  *
32  * <p>
33  * This exception can be thrown by the generated setter methods of the schema
34  * derived Java content classes. However, since fail-fast validation is
35  * an optional feature for JAXB Providers to support, not all setter methods
36  * will throw this exception when a type constraint is violated.
37  *
38  * <p>
39  * If this exception is throw while invoking a fail-fast setter, the value of
40  * the property is guaranteed to remain unchanged, as if the setter were never
41  * called.
42  *
43  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc
44  *         .</li></ul>
45  * @see ValidationEvent
46  * @since JAXB1.0
47  */
48 public class TypeConstraintException extends java.lang.RuntimeException {
49
50     /**
51      * Vendor specific error code
52      *
53      */
54     private String errorCode;
55
56     /**
57      * Exception reference
58      *
59      */
60     private volatile Throwable linkedException;
61
62     static final long serialVersionUID = -3059799699420143848L;
63
64     /**
65      * Construct a TypeConstraintException with the specified detail message. The
66      * errorCode and linkedException will default to null.
```

```
67      *
68      * @param message a description of the exception
69      */
70      public TypeConstraintException(String message) {
71          this( message, null, null );
72      }
73
74      /**
75       * Construct a TypeConstraintException with the specified detail message and vendor
76       * specific errorCode. The linkedException will default to null.
77       *
78       * @param message a description of the exception
79       * @param errorCode a string specifying the vendor specific error code
80       */
81      public TypeConstraintException(String message, String errorCode) {
82          this( message, errorCode, null );
83      }
84
85      /**
86       * Construct a TypeConstraintException with a linkedException. The detail message and
87       * vendor specific errorCode will default to null.
88       *
89       * @param exception the linked exception
90       */
91      public TypeConstraintException(Throwable exception) {
92          this( null, null, exception );
93      }
94
95      /**
96       * Construct a TypeConstraintException with the specified detail message and
97       * linkedException. The errorCode will default to null.
98       *
99       * @param message a description of the exception
100      * @param exception the linked exception
101      */
102      public TypeConstraintException(String message, Throwable exception) {
103          this( message, null, exception );
104      }
105
106      /**
107       * Construct a TypeConstraintException with the specified detail message,
108       * vendor specific errorCode, and linkedException.
109       *
110       * @param message a description of the exception
111       * @param errorCode a string specifying the vendor specific error code
112       * @param exception the linked exception
113       */
114      public TypeConstraintException(String message, String errorCode, Throwable exception) {
115          super( message );
116          this.errorCode = errorCode;
117          this.linkedException = exception;
118      }
119
```

```

120  /**
121   * Get the vendor specific error code
122   *
123   * @return a string specifying the vendor specific error code
124   */
125  public String getErrorCode() {
126      return this.errorCode;
127  }
128
129  /**
130   * Get the linked exception
131   *
132   * @return the linked Exception, null if none exists
133   */
134  public Throwable getLinkedException() {
135      return linkedException;
136  }
137
138  /**
139   * Add a linked Exception.
140   *
141   * @param exception the linked Exception (A null value is permitted and
142   *                 indicates that the linked exception does not exist or
143   *                 is unknown).
144   */
145  public void setLinkedException( Throwable exception ) {
146      this.linkedException = exception;
147  }
148
149  /**
150   * Returns a short description of this TypeConstraintException.
151   *
152   */
153  public String toString() {
154      return linkedException == null ?
155          super.toString() :
156          super.toString() + "\n - with linked exception:\n[" +
157              linkedException.toString()+ "]";
158  }
159
160  /**
161   * Prints this TypeConstraintException and its stack trace (including the stack trace
162   * of the linkedException if it is non-null) to the PrintStream.
163   *
164   * @param s PrintStream to use for output
165   */
166  public void printStackTrace( java.io.PrintStream s ) {
167      if( linkedException != null ) {
168          linkedException.printStackTrace(s);
169          s.println("----- linked to -----");
170      }
171
172      super.printStackTrace(s);

```



```

173     }
174
175     /**
176      * Prints this TypeConstraintException and its stack trace (including the stack trace
177      * of the linkedException if it is non-null) to <tt>System.err</tt>.
178      *
179      */
180     public void printStackTrace() {
181         printStackTrace(System.err);
182     }
183
184 }

```

## 2.24 UnmarshalException.java

Secuencia 26: javax/xml/bind/UnmarshalException.java

```

1  /*
2   * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind;
27
28 /**
29  * This exception indicates that an error has occurred while performing
30  * an unmarshal operation that prevents the JAXB Provider from completing
31  * the operation.
32  *
33  * <p>
34  * The <tt>ValidationEventHandler</tt> can cause this exception to be thrown
35  * during the unmarshal operations. See
36  * {@link ValidationEventHandler#handleEvent(ValidationEvent)}
37  * ValidationEventHandler.handleEvent(ValidationEvent)}.

```

```
38  *
39  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
40  * @see JAXBException
41  * @see Unmarshaller
42  * @see ValidationEventHandler
43  * @since JAXB1.0
44  */
45  public class UnmarshalException extends JAXBException {
46
47      /**
48       * Construct an UnmarshalException with the specified detail message. The
49       * errorCode and linkedException will default to null.
50       *
51       * @param message a description of the exception
52       */
53      public UnmarshalException( String message ) {
54          this( message, null, null );
55      }
56
57      /**
58       * Construct an UnmarshalException with the specified detail message and vendor
59       * specific errorCode. The linkedException will default to null.
60       *
61       * @param message a description of the exception
62       * @param errorCode a string specifying the vendor specific error code
63       */
64      public UnmarshalException( String message, String errorCode ) {
65          this( message, errorCode, null );
66      }
67
68      /**
69       * Construct an UnmarshalException with a linkedException. The detail message and
70       * vendor specific errorCode will default to null.
71       *
72       * @param exception the linked exception
73       */
74      public UnmarshalException( Throwable exception ) {
75          this( null, null, exception );
76      }
77
78      /**
79       * Construct an UnmarshalException with the specified detail message and
80       * linkedException. The errorCode will default to null.
81       *
82       * @param message a description of the exception
83       * @param exception the linked exception
84       */
85      public UnmarshalException( String message, Throwable exception ) {
86          this( message, null, exception );
87      }
88
89      /**
90       * Construct an UnmarshalException with the specified detail message, vendor
```

```

91     * specific errorCode, and linkedException.
92     *
93     * @param message a description of the exception
94     * @param errorCode a string specifying the vendor specific error code
95     * @param exception the linked exception
96     */
97     public UnmarshalException( String message, String errorCode, Throwable exception ) {
98         super( message, errorCode, exception );
99     }
100
101 }

```

## 2.25 Unmarshaller.java

Secuencia 27: javax/xml/bind/Unmarshaller.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import javax.xml.bind.annotation.adapters.XmlAdapter;
29 import javax.xml.bind.attachment.AttachmentUnmarshaller;
30 import javax.xml.validation.Schema;
31 import java.io.Reader;
32
33 /**
34 * The <tt>Unmarshaller</tt> class governs the process of deserializing XML
35 * data into newly created Java content trees, optionally validating the XML
36 * data as it is unmarshalled. It provides an overloading of unmarshal methods
37 * for many different input kinds.
38 *

```

```

39 * <p>
40 * Unmarshalling from a File:
41 * <blockquote>
42 *   <pre>
43 *       JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
44 *       Unmarshaller u = jc.createUnmarshaller();
45 *       Object o = u.unmarshal( new File( "nosferatu.xml" ) );
46 *   </pre>
47 * </blockquote>
48 *
49 *
50 * <p>
51 * Unmarshalling from an InputStream:
52 * <blockquote>
53 *   <pre>
54 *       InputStream is = new FileInputStream( "nosferatu.xml" );
55 *       JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
56 *       Unmarshaller u = jc.createUnmarshaller();
57 *       Object o = u.unmarshal( is );
58 *   </pre>
59 * </blockquote>
60 *
61 * <p>
62 * Unmarshalling from a URL:
63 * <blockquote>
64 *   <pre>
65 *       JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
66 *       Unmarshaller u = jc.createUnmarshaller();
67 *       URL url = new URL( "http://beaker.east/nosferatu.xml" );
68 *       Object o = u.unmarshal( url );
69 *   </pre>
70 * </blockquote>
71 *
72 * <p>
73 * Unmarshalling from a StringBuffer using a
74 * <tt>javax.xml.transform.stream.StreamSource</tt>:
75 * <blockquote>
76 *   <pre>
77 *       JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
78 *       Unmarshaller u = jc.createUnmarshaller();
79 *       StringBuffer xmlStr = new StringBuffer( "<?xml version='1.0'>?>..." );
80 *       Object o = u.unmarshal( new StreamSource( new StringReader( xmlStr.toString() ) ) );
81 *   </pre>
82 * </blockquote>
83 *
84 * <p>
85 * Unmarshalling from a <tt>org.w3c.dom.Node</tt>:
86 * <blockquote>
87 *   <pre>
88 *       JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
89 *       Unmarshaller u = jc.createUnmarshaller();
90 *
91 *       DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

```

```

92 *      dbf.setNamespaceAware(true);
93 *      DocumentBuilder db = dbf.newDocumentBuilder();
94 *      Document doc = db.parse(new File( "nosferatu.xml"));
95
96 *      Object o = u.unmarshal( doc );
97 *      </pre>
98 * </blockquote>
99 *
100 * <p>
101 * Unmarshalling from a <tt>javax.xml.transform.sax.SAXSource</tt> using a
102 * client specified validating SAX2.0 parser:
103 * <blockquote>
104 *      <pre>
105 *          // configure a validating SAX2.0 parser (Xerces2)
106 *          static final String JAXP_SCHEMA_LANGUAGE =
107 *              "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
108 *          static final String JAXP_SCHEMA_LOCATION =
109 *              "http://java.sun.com/xml/jaxp/properties/schemaSource";
110 *          static final String W3C_XML_SCHEMA =
111 *              "http://www.w3.org/2001/XMLSchema";
112 *
113 *          System.setProperty( "javax.xml.parsers.SAXParserFactory",
114 *                              "org.apache.xerces.jaxp.SAXParserFactoryImpl" );
115 *
116 *          SAXParserFactory spf = SAXParserFactory.newInstance();
117 *          spf.setNamespaceAware(true);
118 *          spf.setValidating(true);
119 *          SAXParser saxParser = spf.newSAXParser();
120 *
121 *          try {
122 *              saxParser.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
123 *              saxParser.setProperty(JAXP_SCHEMA_LOCATION, "http://....");
124 *          } catch (SAXNotRecognizedException x) {
125 *              // exception handling omitted
126 *          }
127 *
128 *          XMLReader xmlReader = saxParser.getXMLReader();
129 *          SAXSource source =
130 *              new SAXSource( xmlReader, new InputSource( "http://..." ) );
131 *
132 *          // Setup JAXB to unmarshal
133 *          JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
134 *          Unmarshaller u = jc.createUnmarshaller();
135 *          ValidationEventCollector vec = new ValidationEventCollector();
136 *          u.setEventHandler( vec );
137 *
138 *          // turn off the JAXB provider's default validation mechanism to
139 *          // avoid duplicate validation
140 *          u.setValidating( false )
141 *
142 *          // unmarshal
143 *          Object o = u.unmarshal( source );
144 *

```

```

145 *      // check for events
146 *      if( vec.hasEvents() ) {
147 *          // iterate over events
148 *      }
149 *  </pre>
150 * </blockquote>
151 *
152 * <p>
153 * Unmarshalling from a StAX XMLStreamReader:
154 * <blockquote>
155 *     <pre>
156 *         JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
157 *         Unmarshaller u = jc.createUnmarshaller();
158 *
159 *         javax.xml.stream.XMLStreamReader xmlStreamReader =
160 *             javax.xml.stream.XMLInputFactory().newInstance().createXMLStreamReader( ... );
161 *
162 *         Object o = u.unmarshal( xmlStreamReader );
163 *     </pre>
164 * </blockquote>
165 *
166 * <p>
167 * Unmarshalling from a StAX XMLEventReader:
168 * <blockquote>
169 *     <pre>
170 *         JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
171 *         Unmarshaller u = jc.createUnmarshaller();
172 *
173 *         javax.xml.stream.XMLEventReader xmlEventReader =
174 *             javax.xml.stream.XMLInputFactory().newInstance().createXMLEventReader( ... );
175 *
176 *         Object o = u.unmarshal( xmlEventReader );
177 *     </pre>
178 * </blockquote>
179 *
180 * <p>
181 * <a name="unmarshalEx"></a>
182 * <b>Unmarshalling XML Data</b><br>
183 * <blockquote>
184 * Unmarshalling can deserialize XML data that represents either an entire XML document
185 * or a subtree of an XML document. Typically, it is sufficient to use the
186 * unmarshalling methods described by
187 * <a href="#unmarshalGlobal">Unmarshal root element that is declared globally</a>.
188 * These unmarshal methods utilize {@link JAXBContext}'s mapping of global XML element
189 * declarations and type definitions to JAXB mapped classes to initiate the
190 * unmarshalling of the root element of XML data. When the {@link JAXBContext}'s
191 * mappings are not sufficient to unmarshal the root element of XML data,
192 * the application can assist the unmarshalling process by using the
193 * <a href="#unmarshalByDeclaredType">unmarshal by declaredType methods</a>.
194 * These methods are useful for unmarshalling XML data where
195 * the root element corresponds to a local element declaration in the schema.
196 * </blockquote>
197 *

```

```

198 * <blockquote>
199 * An unmarshal method never returns null. If the unmarshal process is unable to unmarshal
200 * the root of XML content to a JAXB mapped object, a fatal error is reported that
201 * terminates processing by throwing JAXBException.
202 * </blockquote>
203 *
204 * <p>
205 * <a name="unmarshalGlobal"></a>
206 * <b>Unmarshal a root element that is globally declared</b><br>
207 * <blockquote>
208 * The unmarshal methods that do not have an <tt>declaredType</tt> parameter use
209 * {@link JAXBContext} to unmarshal the root element of an XML data. The {@link JAXBContext}
210 * instance is the one that was used to create this <tt>Unmarshaller</tt>. The {@link JAXBContext}
211 * instance maintains a mapping of globally declared XML element and type definition names to
212 * JAXB mapped classes. The unmarshal method checks if {@link JAXBContext} has a mapping
213 * from the root element's XML name and/or <tt>@xsi:type</tt> to a JAXB mapped class. If it does,
214 * it unmarshalls the
215 * XML data using the appropriate JAXB mapped class. Note that when the root element name is
216 * unknown and the root
217 * element has an <tt>@xsi:type</tt>, the XML data is unmarshalled
218 * using that JAXB mapped class as the value of a {@link JAXBElement}.
219 * When the {@link JAXBContext} object does not have a mapping for the root element's name
220 * nor its <tt>@xsi:type</tt>, if it exists,
221 * then the unmarshal operation will abort immediately by throwing a {@link UnmarshalException}
222 * UnmarshalException}. This exception scenario can be worked around by using the unmarshal by
223 * declaredType methods described in the next subsection.
224 * </blockquote>
225 *
226 * <p>
227 * <a name="unmarshalByDeclaredType"></a>
228 * <b>Unmarshal by Declared Type</b><br>
229 * <blockquote>
230 * The unmarshal methods with a <code>declaredType</code> parameter enable an
231 * application to deserialize a root element of XML data, even when
232 * there is no mapping in {@link JAXBContext} of the root element's XML name.
233 * The unmarshaller unmarshals the root element using the application provided
234 * mapping specified as the <tt>declaredType</tt> parameter.
235 * Note that even when the root element's element name is mapped by {@link JAXBContext},
236 * the <code>declaredType</code> parameter overrides that mapping for
237 * deserializing the root element when using these unmarshal methods.
238 * Additionally, when the root element of XML data has an <tt>xsi:type</tt> attribute and
239 * that attribute's value references a type definition that is mapped
240 * to a JAXB mapped class by {@link JAXBContext}, that the root
241 * element's <tt>xsi:type</tt> attribute takes
242 * precedence over the unmarshal methods <tt>declaredType</tt> parameter.
243 * These methods always return a <tt>JAXBElement<declaredType></tt>
244 * instance. The table below shows how the properties of the returned JAXBElement
245 * instance are set.
246 *
247 * <a name="unmarshalDeclaredTypeReturn"></a>
248 * <p>
249 * <table border="2" rules="all" cellpadding="4">
250 * <thead>

```

```

249 *      <tr>
250 *          <th align="center" colspan="2">
251 *              Unmarshal By Declared Type returned JAXBElement
252 *          </tr>
253 *      <tr>
254 *          <th>JAXBElement Property</th>
255 *          <th>Value</th>
256 *      </tr>
257 *      <tr>
258 *          <td>name</td>
259 *          <td><code>xml element name</code></td>
260 *      </tr>
261 *  </thead>
262 *  <tbody>
263 *      <tr>
264 *          <td>value</td>
265 *          <td><code>instanceof declaredType</code></td>
266 *      </tr>
267 *      <tr>
268 *          <td>declaredType</td>
269 *          <td>unmarshal method <code>declaredType</code> parameter</td>
270 *      </tr>
271 *      <tr>
272 *          <td>scope</td>
273 *          <td><code>null</code> <i>(actual scope is unknown)</i></td>
274 *      </tr>
275 *  </tbody>
276 * </table>
277 * </blockquote>
278 *
279 * <p>
280 * The following is an example of
281 * <a href="#unmarshalByDeclaredType">unmarshal by declaredType method</a>.
282 * <p>
283 * Unmarshal by declaredType from a <tt>org.w3c.dom.Node</tt>:
284 * <blockquote>
285 *     <pre>
286 *         Schema fragment for example
287 *         &lt;xs:schema>
288 *             &lt;xs:complexType name="FooType">...&lt;\xs:complexType>
289 *             &lt;!-- global element declaration "PurchaseOrder" -->
290 *             &lt;xs:element name="PurchaseOrder">
291 *                 &lt;xs:complexType>
292 *                     &lt;xs:sequence>
293 *                         &lt;!-- local element declaration "foo" -->
294 *                         &lt;xs:element name="foo" type="FooType"/>
295 *                     ...
296 *                 &lt;/xs:sequence>
297 *             &lt;/xs:complexType>
298 *             &lt;/xs:element>
299 *         &lt;/xs:schema>
300 *     </pre>
301 *
302 *     JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );

```



```

302 *      Unmarshaller u = jc.createUnmarshaller();
303 *
304 *      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
305 *      dbf.setNamespaceAware(true);
306 *      DocumentBuilder db = dbf.newDocumentBuilder();
307 *      Document doc = db.parse(new File( "nosferatu.xml"));
308 *      Element fooSubtree = ...; // traverse DOM till reach xml element foo, constrained by a
309 *                               // local element declaration in schema.
310 *
311 *      // FooType is the JAXB mapping of the type of local element declaration foo.
312 *      JAXBElement<FooType> foo = u.unmarshal( fooSubtree, FooType.class);
313 * </pre>
314 * </blockquote>
315 *
316 * <p>
317 * <b>Support for SAX2.0 Compliant Parsers</b><br>
318 * <blockquote>
319 * A client application has the ability to select the SAX2.0 compliant parser
320 * of their choice. If a SAX parser is not selected, then the JAXB Provider's
321 * default parser will be used. Even though the JAXB Provider's default parser
322 * is not required to be SAX2.0 compliant, all providers are required to allow
323 * a client application to specify their own SAX2.0 parser. Some providers may
324 * require the client application to specify the SAX2.0 parser at schema compile
325 * time. See {@link javax.xml.transform.Source} unmarshal\(Source\)}
326 * for more detail.
327 * </blockquote>
328 *
329 * <p>
330 * <b>Validation and Well-Formedness</b><br>
331 * <blockquote>
332 * <p>
333 * A client application can enable or disable JAXP 1.3 validation
334 * mechanism via the setSchema(javax.xml.validation.Schema) API.
335 * Sophisticated clients can specify their own validating SAX 2.0 compliant
336 * parser and bypass the JAXP 1.3 validation mechanism using the
337 * {@link javax.xml.transform.Source} unmarshal\(Source\)} API.
338 *
339 * <p>
340 * Since unmarshalling invalid XML content is defined in JAXB 2.0,
341 * the Unmarshaller default validation event handler was made more lenient
342 * than in JAXB 1.0. When schema-derived code generated
343 * by JAXB 1.0 binding compiler is registered with {@link JAXBContext},
344 * the default unmarshal validation handler is
345 * {@link javax.xml.bind.helpers.DefaultValidationEventHandler} and it
346 * terminates the marshal operation after encountering either a fatal error or an error.
347 * For a JAXB 2.0 client application, there is no explicitly defined default
348 * validation handler and the default event handling only
349 * terminates the unmarshal operation after encountering a fatal error.
350 *
351 * </blockquote>
352 *
353 * <p>
354 * <a name="supportedProps"></a>

```

```

355 * <b>Supported Properties</b><br>
356 * <blockquote>
357 * <p>
358 * There currently are not any properties required to be supported by all
359 * JAXB Providers on Unmarshaller. However, some providers may support
360 * their own set of provider specific properties.
361 * </blockquote>
362 *
363 * <p>
364 * <a name="unmarshalEventCallback"></a>
365 * <b>Unmarshal Event Callbacks</b><br>
366 * <blockquote>
367 * The {@link Unmarshaller} provides two styles of callback mechanisms
368 * that allow application specific processing during key points in the
369 * unmarshalling process. In 'class defined' event callbacks, application
370 * specific code placed in JAXB mapped classes is triggered during
371 * unmarshalling. 'External listeners' allow for centralized processing
372 * of unmarshal events in one callback method rather than by type event callbacks.
373 * <p>
374 * 'Class defined' event callback methods allow any JAXB mapped class to specify
375 * its own specific callback methods by defining methods with the following method signature:
376 * <blockquote>
377 * <pre>
378 * // This method is called immediately after the object is created and before the unmarshalling
379 * // of this
380 * // object begins. The callback provides an opportunity to initialize JavaBean properties prior
381 * // to unmarshalling.
382 * void beforeUnmarshal(Unmarshaller, Object parent);
383 *
384 * //This method is called after all the properties (except IDREF) are unmarshalled for this
385 * //object,
386 * //but before this object is set to the parent object.
387 * void afterUnmarshal(Unmarshaller, Object parent);
388 * </pre>
389 * </blockquote>
390 * The class defined callback methods should be used when the callback method requires
391 * access to non-public methods and/or fields of the class.
392 * <p>
393 * The external listener callback mechanism enables the registration of a {@link Listener}
394 * instance with an {@link Unmarshaller#setListener(Listener)}. The external listener receives all
395 * callback events,
396 * allowing for more centralized processing than per class defined callback methods. The external
397 * listener
398 * receives events when unmarshalling process is marshalling to a JAXB element or to JAXB mapped
399 * class.
400 * <p>
401 * The 'class defined' and external listener event callback methods are independent of each other,
402 * both can be called for one event. The invocation ordering when both listener callback methods
403 * exist is
404 * defined in {@link Listener#beforeUnmarshal(Object, Object)} and {@link Listener#afterUnmarshal(
405 * Object, Object)}.
406 * <p>
407 * An event callback method throwing an exception terminates the current unmarshal process.

```

```

400 *
401 * </blockquote>
402 *
403 * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
404   Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
405 * @see JAXBContext
406 * @see Marshaller
407 * @see Validator
408 * @since JAXB1.0
409 */
409 public interface Unmarshaller {
410
411   /**
412    * Unmarshal XML data from the specified file and return the resulting
413    * content tree.
414    *
415    * <p>
416    * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
417    *
418    * @param f the file to unmarshal XML data from
419    * @return the newly created root object of the java content tree
420    *
421    * @throws JAXBException
422    *   If any unexpected errors occur while unmarshalling
423    * @throws UnmarshalException
424    *   If the {@link ValidationEventHandler ValidationEventHandler}
425    *   returns false from its <tt>handleEvent</tt> method or the
426    *   <tt>Unmarshaller</tt> is unable to perform the XML to Java
427    *   binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
428    * @throws IllegalArgumentException
429    *   If the file parameter is null
430    */
431   public Object unmarshal( java.io.File f ) throws JAXBException;
432
433   /**
434    * Unmarshal XML data from the specified InputStream and return the
435    * resulting content tree. Validation event location information may
436    * be incomplete when using this form of the unmarshal API.
437    *
438    * <p>
439    * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
440    *
441    * @param is the InputStream to unmarshal XML data from
442    * @return the newly created root object of the java content tree
443    *
444    * @throws JAXBException
445    *   If any unexpected errors occur while unmarshalling
446    * @throws UnmarshalException
447    *   If the {@link ValidationEventHandler ValidationEventHandler}
448    *   returns false from its <tt>handleEvent</tt> method or the
449    *   <tt>Unmarshaller</tt> is unable to perform the XML to Java
450    *   binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
451    * @throws IllegalArgumentException

```

```

452     *      If the InputStream parameter is null
453     */
454     public Object unmarshal( java.io.InputStream is ) throws JAXBException;
455
456     /**
457     * Unmarshal XML data from the specified Reader and return the
458     * resulting content tree. Validation event location information may
459     * be incomplete when using this form of the unmarshal API,
460     * because a Reader does not provide the system ID.
461     *
462     * <p>
463     * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
464     *
465     * @param reader the Reader to unmarshal XML data from
466     * @return the newly created root object of the java content tree
467     *
468     * @throws JAXBException
469     *      If any unexpected errors occur while unmarshalling
470     * @throws UnmarshalException
471     *      If the {@link ValidationEventHandler ValidationEventHandler}
472     *      returns false from its <tt>handleEvent</tt> method or the
473     *      <tt>Unmarshaller</tt> is unable to perform the XML to Java
474     *      binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
475     * @throws IllegalArgumentException
476     *      If the InputStream parameter is null
477     * @since JAXB2.0
478     */
479     public Object unmarshal( Reader reader ) throws JAXBException;
480
481     /**
482     * Unmarshal XML data from the specified URL and return the resulting
483     * content tree.
484     *
485     * <p>
486     * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
487     *
488     * @param url the url to unmarshal XML data from
489     * @return the newly created root object of the java content tree
490     *
491     * @throws JAXBException
492     *      If any unexpected errors occur while unmarshalling
493     * @throws UnmarshalException
494     *      If the {@link ValidationEventHandler ValidationEventHandler}
495     *      returns false from its <tt>handleEvent</tt> method or the
496     *      <tt>Unmarshaller</tt> is unable to perform the XML to Java
497     *      binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
498     * @throws IllegalArgumentException
499     *      If the URL parameter is null
500     */
501     public Object unmarshal( java.net.URL url ) throws JAXBException;
502
503     /**
504     * Unmarshal XML data from the specified SAX InputSource and return the

```

```

505     * resulting content tree.
506     *
507     * <p>
508     * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
509     *
510     * @param source the input source to unmarshal XML data from
511     * @return the newly created root object of the java content tree
512     *
513     * @throws JAXBException
514     *     If any unexpected errors occur while unmarshalling
515     * @throws UnmarshalException
516     *     If the {@link ValidationEventHandler ValidationEventHandler}
517     *     returns false from its <tt>handleEvent</tt> method or the
518     *     <tt>Unmarshaller</tt> is unable to perform the XML to Java
519     *     binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
520     * @throws IllegalArgumentException
521     *     If the InputSource parameter is null
522     */
523     public Object unmarshal( org.xml.sax.InputSource source ) throws JAXBException;
524
525     /**
526     * Unmarshal global XML data from the specified DOM tree and return the resulting
527     * content tree.
528     *
529     * <p>
530     * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
531     *
532     * @param node
533     *     the document/element to unmarshal XML data from.
534     *     The caller must support at least Document and Element.
535     * @return the newly created root object of the java content tree
536     *
537     * @throws JAXBException
538     *     If any unexpected errors occur while unmarshalling
539     * @throws UnmarshalException
540     *     If the {@link ValidationEventHandler ValidationEventHandler}
541     *     returns false from its <tt>handleEvent</tt> method or the
542     *     <tt>Unmarshaller</tt> is unable to perform the XML to Java
543     *     binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
544     * @throws IllegalArgumentException
545     *     If the Node parameter is null
546     * @see #unmarshal(org.w3c.dom.Node, Class)
547     */
548     public Object unmarshal( org.w3c.dom.Node node ) throws JAXBException;
549
550     /**
551     * Unmarshal XML data by JAXB mapped <tt>declaredType</tt>
552     * and return the resulting content tree.
553     *
554     * <p>
555     * Implements <a href="#unmarshalByDeclaredType">Unmarshal by Declared Type</a>
556     *
557     * @param node

```

```

558     * the document/element to unmarshal XML data from.
559     * The caller must support at least Document and Element.
560     * @param declaredType
561     * appropriate JAXB mapped class to hold <tt>node</tt>'s XML data.
562     *
563     * @return <a href="#unmarshalDeclaredTypeReturn">JAXB Element</a> representation of <tt>node</
564         tt>
565     *
566     * @throws JAXBException
567     * If any unexpected errors occur while unmarshalling
568     * @throws UnmarshalException
569     * If the {@link ValidationEventHandler ValidationEventHandler}
570     * returns false from its <tt>handleEvent</tt> method or the
571     * <tt>Unmarshaller</tt> is unable to perform the XML to Java
572     * binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
573     * @throws IllegalArgumentException
574     * If any parameter is null
575     * @since JAXB2.0
576     */
577     public <T> JAXBElement<T> unmarshal( org.w3c.dom.Node node, Class<T> declaredType ) throws
578         JAXBException;
579
580 /**
581  * Unmarshal XML data from the specified XML Source and return the
582  * resulting content tree.
583  *
584  * <p>
585  * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
586  *
587  * <p>
588  * <a name="saxParserPluggable"></a>
589  * <b>SAX 2.0 Parser Pluggability</b>
590  *
591  * <p>
592  * A client application can choose not to use the default parser mechanism
593  * supplied with their JAXB provider. Any SAX 2.0 compliant parser can be
594  * substituted for the JAXB provider's default mechanism. To do so, the
595  * client application must properly configure a <tt>SAXSource</tt> containing
596  * an <tt>XMLReader</tt> implemented by the SAX 2.0 parser provider. If the
597  * <tt>XMLReader</tt> has an <tt>org.xml.sax.ErrorHandler</tt> registered
598  * on it, it will be replaced by the JAXB Provider so that validation errors
599  * can be reported via the <tt>ValidationEventHandler</tt> mechanism of
600  * JAXB. If the <tt>SAXSource</tt> does not contain an <tt>XMLReader</tt>,
601  * then the JAXB provider's default parser mechanism will be used.
602  *
603  * <p>
604  * This parser replacement mechanism can also be used to replace the JAXB
605  * provider's unmarshal-time validation engine. The client application
606  * must properly configure their SAX 2.0 compliant parser to perform
607  * validation (as shown in the example above). Any <tt>SAXParserExceptions
608  * </tt> encountered by the parser during the unmarshal operation will be
609  * processed by the JAXB provider and converted into JAXB
610  * <tt>ValidationEvent</tt> objects which will be reported back to the
611  * client via the <tt>ValidationEventHandler</tt> registered with the
612  * <tt>Unmarshaller</tt>. <i>Note:</i> specifying a substitute validating

```

```

609 * SAX 2.0 parser for unmarshalling does not necessarily replace the
610 * validation engine used by the JAXB provider for performing on-demand
611 * validation.
612 * <p>
613 * The only way for a client application to specify an alternate parser
614 * mechanism to be used during unmarshal is via the
615 * <tt>unmarshal(SAXSource)</tt> API. All other forms of the unmarshal
616 * method (File, URL, Node, etc) will use the JAXB provider's default
617 * parser and validator mechanisms.
618 *
619 * @param source the XML Source to unmarshal XML data from (providers are
620 * only required to support SAXSource, DOMSource, and StreamSource)
621 * @return the newly created root object of the java content tree
622 *
623 * @throws JAXBException
624 * If any unexpected errors occur while unmarshalling
625 * @throws UnmarshalException
626 * If the {@link ValidationEventHandler ValidationEventHandler}
627 * returns false from its <tt>handleEvent</tt> method or the
628 * <tt>Unmarshaller</tt> is unable to perform the XML to Java
629 * binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
630 * @throws IllegalArgumentException
631 * If the Source parameter is null
632 * @see #unmarshal(javax.xml.transform.Source, Class)
633 */
634 public Object unmarshal( javax.xml.transform.Source source )
635     throws JAXBException;
636
637
638 /**
639 * Unmarshal XML data from the specified XML Source by <tt>declaredType</tt> and return the
640 * resulting content tree.
641 *
642 * <p>
643 * Implements <a href="#unmarshalByDeclaredType">Unmarshal by Declared Type</a>
644 *
645 * <p>
646 * See <a href="#saxParserPluggable">SAX 2.0 Parser Pluggability</a>
647 *
648 * @param source the XML Source to unmarshal XML data from (providers are
649 * only required to support SAXSource, DOMSource, and StreamSource)
650 * @param declaredType
651 * appropriate JAXB mapped class to hold <tt>source</tt>'s xml root element
652 * @return Java content rooted by <a href="#unmarshalDeclaredTypeReturn">JAXB Element</a>
653 *
654 * @throws JAXBException
655 * If any unexpected errors occur while unmarshalling
656 * @throws UnmarshalException
657 * If the {@link ValidationEventHandler ValidationEventHandler}
658 * returns false from its <tt>handleEvent</tt> method or the
659 * <tt>Unmarshaller</tt> is unable to perform the XML to Java
660 * binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
661 * @throws IllegalArgumentException

```



```

662     *      If any parameter is null
663     * @since JAXB2.0
664     */
665     public <T> JAXBElement<T> unmarshal( javax.xml.transform.Source source, Class<T> declaredType )
666         throws JAXBException;
667
668     /**
669     * Unmarshal XML data from the specified pull parser and return the
670     * resulting content tree.
671     *
672     * <p>
673     * Implements <a href="#unmarshalGlobal">Unmarshal Global Root Element</a>.
674     *
675     * <p>
676     * This method assumes that the parser is on a START_DOCUMENT or
677     * START_ELEMENT event. Unmarshalling will be done from this
678     * start event to the corresponding end event. If this method
679     * returns successfully, the <tt>reader</tt> will be pointing at
680     * the token right after the end event.
681     *
682     * @param reader
683     *      The parser to be read.
684     * @return
685     *      the newly created root object of the java content tree.
686     *
687     * @throws JAXBException
688     *      If any unexpected errors occur while unmarshalling
689     * @throws UnmarshalException
690     *      If the {@link ValidationEventHandler ValidationEventHandler}
691     *      returns false from its <tt>handleEvent</tt> method or the
692     *      <tt>Unmarshaller</tt> is unable to perform the XML to Java
693     *      binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
694     * @throws IllegalArgumentException
695     *      If the <tt>reader</tt> parameter is null
696     * @throws IllegalStateException
697     *      If <tt>reader</tt> is not pointing to a START_DOCUMENT or
698     *      START_ELEMENT event.
699     * @since JAXB2.0
700     * @see #unmarshal(javax.xml.stream.XMLStreamReader, Class)
701     */
702     public Object unmarshal( javax.xml.stream.XMLStreamReader reader )
703         throws JAXBException;
704
705     /**
706     * Unmarshal root element to JAXB mapped <tt>declaredType</tt>
707     * and return the resulting content tree.
708     *
709     * <p>
710     * This method implements <a href="#unmarshalByDeclaredType">unmarshal by declaredType</a>.
711     * <p>
712     * This method assumes that the parser is on a START_DOCUMENT or
713     * START_ELEMENT event. Unmarshalling will be done from this
714     * start event to the corresponding end event. If this method

```



```

715 * returns successfully, the <tt>reader</tt> will be pointing at
716 * the token right after the end event.
717 *
718 * @param reader
719 *     The parser to be read.
720 * @param declaredType
721 *     appropriate JAXB mapped class to hold <tt>reader</tt>'s START_ELEMENT XML data.
722 *
723 * @return content tree rooted by <a href="#unmarshalDeclaredTypeReturn">JAXB Element
       representation</a>
724 *
725 * @throws JAXBException
726 *     If any unexpected errors occur while unmarshalling
727 * @throws UnmarshalException
728 *     If the {@link ValidationEventHandler ValidationEventHandler}
729 *     returns false from its <tt>handleEvent</tt> method or the
730 *     <tt>Unmarshaller</tt> is unable to perform the XML to Java
731 *     binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
732 * @throws IllegalArgumentException
733 *     If any parameter is null
734 * @since JAXB2.0
735 */
736 public <T> JAXBElement<T> unmarshal( javax.xml.stream.XMLStreamReader reader, Class<T>
       declaredType ) throws JAXBException;
737
738 /**
739 * Unmarshal XML data from the specified pull parser and return the
740 * resulting content tree.
741 *
742 * <p>
743 * This method is an <a href="#unmarshalGlobal">Unmarshal Global Root method</a>.
744 *
745 * <p>
746 * This method assumes that the parser is on a START_DOCUMENT or
747 * START_ELEMENT event. Unmarshalling will be done from this
748 * start event to the corresponding end event. If this method
749 * returns successfully, the <tt>reader</tt> will be pointing at
750 * the token right after the end event.
751 *
752 * @param reader
753 *     The parser to be read.
754 * @return
755 *     the newly created root object of the java content tree.
756 *
757 * @throws JAXBException
758 *     If any unexpected errors occur while unmarshalling
759 * @throws UnmarshalException
760 *     If the {@link ValidationEventHandler ValidationEventHandler}
761 *     returns false from its <tt>handleEvent</tt> method or the
762 *     <tt>Unmarshaller</tt> is unable to perform the XML to Java
763 *     binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
764 * @throws IllegalArgumentException
765 *     If the <tt>reader</tt> parameter is null

```

```

766     * @throws IllegalStateException
767     *     If <tt>reader</tt> is not pointing to a START_DOCUMENT or
768     *     START_ELEMENT event.
769     * @since JAXB2.0
770     * @see #unmarshal(javax.xml.stream.XMLStreamReader, Class)
771     */
772     public Object unmarshal( javax.xml.stream.XMLStreamReader reader )
773         throws JAXBException;
774
775     /**
776     * Unmarshal root element to JAXB mapped <tt>declaredType</tt>
777     * and return the resulting content tree.
778     *
779     * <p>
780     * This method implements <a href="#unmarshalByDeclaredType">unmarshal by declaredType</a>.
781     *
782     * <p>
783     * This method assumes that the parser is on a START_DOCUMENT or
784     * START_ELEMENT event. Unmarshalling will be done from this
785     * start event to the corresponding end event. If this method
786     * returns successfully, the <tt>reader</tt> will be pointing at
787     * the token right after the end event.
788     *
789     * @param reader
790     *     The parser to be read.
791     * @param declaredType
792     *     appropriate JAXB mapped class to hold <tt>reader</tt>'s START_ELEMENT XML data.
793     *
794     * @return    content tree rooted by <a href="#unmarshalDeclaredTypeReturn">JAXB Element
795     *     representation</a>
796     *
797     * @throws JAXBException
798     *     If any unexpected errors occur while unmarshalling
799     * @throws UnmarshalException
800     *     If the {@link ValidationEventHandler ValidationEventHandler}
801     *     returns false from its <tt>handleEvent</tt> method or the
802     *     <tt>Unmarshaller</tt> is unable to perform the XML to Java
803     *     binding. See <a href="#unmarshalEx">Unmarshalling XML Data</a>
804     * @throws IllegalArgumentException
805     *     If any parameter is null
806     * @since JAXB2.0
807     */
808     public <T> JAXBElement<T> unmarshal( javax.xml.stream.XMLStreamReader reader, Class<T>
809         declaredType ) throws JAXBException;
810
811     /**
812     * Get an unmarshaller handler object that can be used as a component in
813     * an XML pipeline.
814     *
815     * <p>
816     * The JAXB Provider can return the same handler object for multiple
817     * invocations of this method. In other words, this method does not
818     * necessarily create a new instance of <tt>UnmarshallerHandler</tt>. If the

```

```

817     * application needs to use more than one <tt>UnmarshallerHandler</tt>, it
818     * should create more than one <tt>Unmarshaller</tt>.
819     *
820     * @return the unmarshaller handler object
821     * @see UnmarshallerHandler
822     */
823     public UnmarshallerHandler getUnmarshallerHandler();
824
825     /**
826     * Specifies whether or not the default validation mechanism of the
827     * <tt>Unmarshaller</tt> should validate during unmarshal operations.
828     * By default, the <tt>Unmarshaller</tt> does not validate.
829     * <p>
830     * This method may only be invoked before or after calling one of the
831     * unmarshal methods.
832     * <p>
833     * This method only controls the JAXB Provider's default unmarshal-time
834     * validation mechanism - it has no impact on clients that specify their
835     * own validating SAX 2.0 compliant parser. Clients that specify their
836     * own unmarshal-time validation mechanism may wish to turn off the JAXB
837     * Provider's default validation mechanism via this API to avoid "double
838     * validation".
839     * <p>
840     * This method is deprecated as of JAXB 2.0 - please use the new
841     * {@link #setSchema(javax.xml.validation.Schema)} API.
842     *
843     * @param validating true if the Unmarshaller should validate during
844     *     unmarshal, false otherwise
845     * @throws JAXBException if an error occurred while enabling or disabling
846     *     validation at unmarshal time
847     * @throws UnsupportedOperationException could be thrown if this method is
848     *     invoked on an Unmarshaller created from a JAXBContext referencing
849     *     JAXB 2.0 mapped classes
850     * @deprecated since JAXB2.0, please see {@link #setSchema(javax.xml.validation.Schema)}
851     */
852     public void setValidating( boolean validating )
853         throws JAXBException;
854
855     /**
856     * Indicates whether or not the <tt>Unmarshaller</tt> is configured to
857     * validate during unmarshal operations.
858     * <p>
859     * This API returns the state of the JAXB Provider's default unmarshal-time
860     * validation mechanism.
861     * <p>
862     * This method is deprecated as of JAXB 2.0 - please use the new
863     * {@link #getSchema()} API.
864     *
865     * @return true if the Unmarshaller is configured to validate during
866     *     unmarshal operations, false otherwise
867     * @throws JAXBException if an error occurs while retrieving the validating
868     *     flag
869     * @throws UnsupportedOperationException could be thrown if this method is

```

```

870     *      invoked on an Unmarshaller created from a JAXBContext referencing
871     *      JAXB 2.0 mapped classes
872     * @deprecated since JAXB2.0, please see {@link #getSchema()}
873     */
874     public boolean isValidating()
875         throws JAXBException;
876
877     /**
878     * Allow an application to register a <tt>ValidationEventHandler</tt>.
879     * <p>
880     * The <tt>ValidationEventHandler</tt> will be called by the JAXB Provider
881     * if any validation errors are encountered during calls to any of the
882     * unmarshal methods. If the client application does not register a
883     * <tt>ValidationEventHandler</tt> before invoking the unmarshal methods,
884     * then <tt>ValidationEvents</tt> will be handled by the default event
885     * handler which will terminate the unmarshal operation after the first
886     * error or fatal error is encountered.
887     * <p>
888     * Calling this method with a null parameter will cause the Unmarshaller
889     * to revert back to the default event handler.
890     *
891     * @param handler the validation event handler
892     * @throws JAXBException if an error was encountered while setting the
893     *         event handler
894     */
895     public void setEventHandler( ValidationEventHandler handler )
896         throws JAXBException;
897
898     /**
899     * Return the current event handler or the default event handler if one
900     * hasn't been set.
901     *
902     * @return the current ValidationEventHandler or the default event handler
903     *         if it hasn't been set
904     * @throws JAXBException if an error was encountered while getting the
905     *         current event handler
906     */
907     public ValidationEventHandler getEventHandler()
908         throws JAXBException;
909
910     /**
911     * Set the particular property in the underlying implementation of
912     * <tt>Unmarshaller</tt>. This method can only be used to set one of
913     * the standard JAXB defined properties above or a provider specific
914     * property. Attempting to set an undefined property will result in
915     * a PropertyException being thrown. See <a href="#supportedProps">
916     * Supported Properties</a>.
917     *
918     * @param name the name of the property to be set. This value can either
919     *         be specified using one of the constant fields or a user
920     *         supplied string.
921     * @param value the value of the property to be set
922     *

```

```

923     * @throws PropertyException when there is an error processing the given
924     *           property or value
925     * @throws IllegalArgumentException
926     *       If the name parameter is null
927     */
928     public void setProperty( String name, Object value )
929         throws PropertyException;
930
931     /**
932     * Get the particular property in the underlying implementation of
933     * <tt>Unmarshaller</tt>. This method can only be used to get one of
934     * the standard JAXB defined properties above or a provider specific
935     * property. Attempting to get an undefined property will result in
936     * a PropertyException being thrown. See <a href="#supportedProps">
937     * Supported Properties</a>.
938     *
939     * @param name the name of the property to retrieve
940     * @return the value of the requested property
941     *
942     * @throws PropertyException
943     *       when there is an error retrieving the given property or value
944     *       property name
945     * @throws IllegalArgumentException
946     *       If the name parameter is null
947     */
948     public Object getProperty( String name ) throws PropertyException;
949
950     /**
951     * Specify the JAXP 1.3 {@link javax.xml.validation.Schema Schema}
952     * object that should be used to validate subsequent unmarshal operations
953     * against. Passing null into this method will disable validation.
954     * <p>
955     * This method replaces the deprecated {@link #setValidating(boolean) setValidating(boolean)}
956     * API.
957     *
958     * <p>
959     * Initially this property is set to <tt>null</tt>.
960     *
961     * @param schema Schema object to validate unmarshal operations against or null to disable
962     *       validation
963     * @throws UnsupportedOperationException could be thrown if this method is
964     *       invoked on an Unmarshaller created from a JAXBContext referencing
965     *       JAXB 1.0 mapped classes
966     * @since JAXB2.0
967     */
968
969     public void setSchema( javax.xml.validation.Schema schema );
970
971     /**
972     * Get the JAXP 1.3 {@link javax.xml.validation.Schema Schema} object
973     * being used to perform unmarshal-time validation. If there is no
974     * Schema set on the unmarshaller, then this method will return null
975     * indicating that unmarshal-time validation will not be performed.
976     * <p>

```

```

975     * This method provides replacement functionality for the deprecated
976     * {@link #isValidating()} API as well as access to the Schema object.
977     * To determine if the Unmarshaller has validation enabled, simply
978     * test the return type for null:
979     * <p>
980     * <code>
981     *     boolean isValidating = u.getSchema()!=null;
982     * </code>
983     *
984     * @return the Schema object being used to perform unmarshal-time
985     *         validation or null if not present
986     * @throws UnsupportedOperationException could be thrown if this method is
987     *         invoked on an Unmarshaller created from a JAXBContext referencing
988     *         JAXB 1.0 mapped classes
989     * @since JAXB2.0
990     */
991     public javax.xml.validation.Schema getSchema();
992
993     /**
994     * Associates a configured instance of {@link XmlAdapter} with this unmarshaller.
995     *
996     * <p>
997     * This is a convenience method that invokes <code>setAdapter(adapter.getClass(),adapter);</code>.
998     *
999     * @see #setAdapter(Class,XmlAdapter)
1000    * @throws IllegalArgumentException
1001    *         if the adapter parameter is null.
1002    * @throws UnsupportedOperationException
1003    *         if invoked against a JAXB 1.0 implementation.
1004    * @since JAXB2.0
1005    */
1006    public void setAdapter( XmlAdapter adapter );
1007
1008    /**
1009    * Associates a configured instance of {@link XmlAdapter} with this unmarshaller.
1010    *
1011    * <p>
1012    * Every unmarshaller internally maintains a
1013    * {@link java.util.Map}<{@link Class},{@link XmlAdapter}>,
1014    * which it uses for unmarshalling classes whose fields/methods are annotated
1015    * with {@link javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter}.
1016    *
1017    * <p>
1018    * This method allows applications to use a configured instance of {@link XmlAdapter}.
1019    * When an instance of an adapter is not given, an unmarshaller will create
1020    * one by invoking its default constructor.
1021    *
1022    * @param type
1023    *         The type of the adapter. The specified instance will be used when
1024    *         {@link javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter#value()}
1025    *         refers to this type.
1026    * @param adapter

```

```

1027 *      The instance of the adapter to be used. If null, it will un-register
1028 *      the current adapter set for this type.
1029 * @throws IllegalArgumentException
1030 *      if the type parameter is null.
1031 * @throws UnsupportedOperationException
1032 *      if invoked against a JAXB 1.0 implementation.
1033 * @since JAXB2.0
1034 */
1035 public <A extends XmlAdapter> void setAdapter( Class<A> type, A adapter );
1036
1037 /**
1038 * Gets the adapter associated with the specified type.
1039 *
1040 * This is the reverse operation of the {@link #setAdapter} method.
1041 *
1042 * @throws IllegalArgumentException
1043 *      if the type parameter is null.
1044 * @throws UnsupportedOperationException
1045 *      if invoked against a JAXB 1.0 implementation.
1046 * @since JAXB2.0
1047 */
1048 public <A extends XmlAdapter> A getAdapter( Class<A> type );
1049
1050 /**
1051 * <p>Associate a context that resolves cid's, content-id URIs, to
1052 * binary data passed as attachments.</p>
1053 * <p>
1054 * <p>Unmarshal time validation, enabled via {@link #setSchema(Schema)},
1055 * must be supported even when unmarshaller is performing XOP processing.
1056 * </p>
1057 *
1058 * @throws IllegalStateException if attempt to concurrently call this
1059 *      method during a unmarshal operation.
1060 */
1061 void setAttachmentUnmarshaller(AttachmentUnmarshaller au);
1062
1063 AttachmentUnmarshaller getAttachmentUnmarshaller();
1064
1065 /**
1066 * <p>
1067 * Register an instance of an implementation of this class with {@link Unmarshaller} to
1068 *      externally listen
1069 * for unmarshal events.
1070 * <p>
1071 * This class enables pre and post processing of an instance of a JAXB mapped class
1072 * as XML data is unmarshalled into it. The event callbacks are called when unmarshalling
1073 * XML content into a JAXBElement instance or a JAXB mapped class that represents a complex
1074 *      type definition.
1075 * The event callbacks are not called when unmarshalling to an instance of a
1076 * Java datatype that represents a simple type definition.
1077 * <p>
1078 * <p>

```

```

1078 * External listener is one of two different mechanisms for defining unmarshal event callbacks.
1079 * See <a href="Unmarshaller.html#unmarshalEventCallback">Unmarshal Event Callbacks</a> for an
    overview.
1080 * <p/>
1081 * (@link #setListener(Listener))
1082 * (@link #getListener())
1083 *
1084 * @since JAXB2.0
1085 */
1086 public static abstract class Listener {
1087     /**
1088      * <p/>
1089      * Callback method invoked before unmarshalling into <tt>target</tt>.
1090      * <p/>
1091      * <p/>
1092      * This method is invoked immediately after <tt>target</tt> was created and
1093      * before the unmarshalling of this object begins. Note that
1094      * if the class of <tt>target</tt> defines its own <tt>beforeUnmarshal</tt> method,
1095      * the class specific callback method is invoked before this method is invoked.
1096      *
1097      * @param target non-null instance of JAXB mapped class prior to unmarshalling into it.
1098      * @param parent instance of JAXB mapped class that will eventually reference <tt>target</
1099      *         <tt>null</tt> when <tt>target</tt> is root element.
1100      */
1101     public void beforeUnmarshal(Object target, Object parent) {
1102     }
1103
1104     /**
1105      * <p/>
1106      * Callback method invoked after unmarshalling XML data into <tt>target</tt>.
1107      * <p/>
1108      * <p/>
1109      * This method is invoked after all the properties (except IDREF) are unmarshalled into <tt
1110      * >target</tt>,
1111      * but before <tt>target</tt> is set into its <tt>parent</tt> object.
1112      * Note that if the class of <tt>target</tt> defines its own <tt>afterUnmarshal</tt> method
1113      * ,
1114      * the class specific callback method is invoked before this method is invoked.
1115      *
1116      * @param target non-null instance of JAXB mapped class prior to unmarshalling into it.
1117      * @param parent instance of JAXB mapped class that will reference <tt>target</tt>.
1118      *         <tt>null</tt> when <tt>target</tt> is root element.
1119      */
1120     public void afterUnmarshal(Object target, Object parent) {
1121     }
1122 }
1123
1124 /**
1125  * <p>
1126  * Register unmarshal event callback {@link Listener} with this {@link Unmarshaller}.
1127  *
1128  * <p>

```



```

1127     * There is only one Listener per Unmarshaller. Setting a Listener replaces the previous set
1128     * Listener.
1129     * One can unregister current Listener by setting listener to <tt>null</tt>.
1130     *
1131     * @param listener provides unmarshal event callbacks for this {@link Unmarshaller}
1132     * @since JAXB2.0
1133     */
1134     public void setListener(Listener listener);
1135
1136     /**
1137     * <p>Return {@link Listener} registered with this {@link Unmarshaller}.
1138     *
1139     * @return registered {@link Listener} or <code>null</code> if no Listener is registered with
1140     *         this Unmarshaller.
1141     * @since JAXB2.0
1142     */
1143     public Listener getListener();
1144 }

```

## 2.26 UnmarshallerHandler.java

Secuencia 28: javax/xml/bind/UnmarshallerHandler.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 import org.xml.sax.ContentHandler;
29
30 /**
31  * Unmarshaller implemented as SAX ContentHandler.

```

```

32 *
33 * <p>
34 * Applications can use this interface to use their JAXB provider as a component
35 * in an XML pipeline. For example:
36 *
37 * <pre>
38 *     JAXBContext context = JAXBContext.newInstance( "org.acme.foo" );
39 *
40 *     Unmarshaller unmarshaller = context.createUnmarshaller();
41 *
42 *     UnmarshallerHandler unmarshallerHandler = unmarshaller.getUnmarshallerHandler();
43 *
44 *     SAXParserFactory spf = SAXParserFactory.newInstance();
45 *     spf.setNamespaceAware( true );
46 *
47 *     XMLReader xmlReader = spf.newSAXParser().getXMLReader();
48 *     xmlReader.setContentHandler( unmarshallerHandler );
49 *     xmlReader.parse(new InputSource( new FileInputStream( XML_FILE ) ) );
50 *
51 *     MyObject myObject= (MyObject)unmarshallerHandler.getResult();
52 * </pre>
53 *
54 * <p>
55 * This interface is reusable: even if the user fails to unmarshal
56 * an object, s/he can still start a new round of unmarshalling.
57 *
58 * @author <ul><li>Kohsuke KAWAGUCHI, Sun Microsystems, Inc.</li></ul>
59 * @see Unmarshaller#getUnmarshallerHandler()
60 * @since JAXB1.0
61 */
62 public interface UnmarshallerHandler extends ContentHandler
63 {
64     /**
65      * Obtains the unmarshalled result.
66      *
67      * This method can be called only after this handler
68      * receives the endDocument SAX event.
69      *
70      * @exception IllegalStateException
71      *         if this method is called before this handler
72      *         receives the endDocument event.
73      *
74      * @exception JAXBException
75      *         if there is any unmarshalling error.
76      *         Note that the implementation is allowed to throw SAXException
77      *         during the parsing when it finds an error.
78      *
79      * @return
80      *         always return a non-null valid object which was unmarshalled.
81      */
82     Object getResult() throws JAXBException, IllegalStateException;
83 }

```

**2.27** ValidationEvent.java

Secuencia 29: javax/xml/bind/ValidationEvent.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 /**
29  * This event indicates that a problem was encountered while validating the
30  * incoming XML data during an unmarshal operation, while performing
31  * on-demand validation of the Java content tree, or while marshalling the
32  * Java content tree back to XML data.
33  *
34  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
35  * Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
36  * @see Validator
37  * @see ValidationEventHandler
38  * @since JAXB1.0
39  */
40 public interface ValidationEvent {
41     /**
42      * Conditions that are not errors or fatal errors as defined by the
43      * XML 1.0 recommendation
44      */
45     public static final int WARNING = 0;
46
47     /**
48      * Conditions that correspond to the definition of "error" in section
49      * 1.2 of the W3C XML 1.0 Recommendation

```

```

50     */
51     public static final int ERROR          = 1;
52
53     /**
54      * Conditions that correspond to the definition of "fatal error" in section
55      * 1.2 of the W3C XML 1.0 Recommendation
56      */
57     public static final int FATAL_ERROR = 2;
58
59     /**
60      * Retrieve the severity code for this warning/error.
61      *
62      * <p>
63      * Must be one of <tt>ValidationError.WARNING</tt>,
64      * <tt>ValidationError.ERROR</tt>, or <tt>ValidationError.FATAL_ERROR</tt>.
65      *
66      * @return the severity code for this warning/error
67      */
68     public int getSeverity();
69
70     /**
71      * Retrieve the text message for this warning/error.
72      *
73      * @return the text message for this warning/error or null if one wasn't set
74      */
75     public String getMessage();
76
77     /**
78      * Retrieve the linked exception for this warning/error.
79      *
80      * @return the linked exception for this warning/error or null if one
81      *         wasn't set
82      */
83     public Throwable getLinkedException();
84
85     /**
86      * Retrieve the locator for this warning/error.
87      *
88      * @return the locator that indicates where the warning/error occurred
89      */
90     public ValidationEventLocator getLocator();
91
92 }

```

## 2.28 ValidationEventHandler.java

Secuencia 30: javax/xml/bind/ValidationEventHandler.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *

```

```

7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 /**
29  * A basic event handler interface for validation errors.
30  *
31  * <p>
32  * If an application needs to implement customized event handling, it must
33  * implement this interface and then register it with either the
34  * {@link Unmarshaller#setEventHandler(ValidationEventHandler) Unmarshaller},
35  * the {@link Validator#setEventHandler(ValidationEventHandler) Validator}, or
36  * the {@link Marshaller#setEventHandler(ValidationEventHandler) Marshaller}.
37  * The JAXB Provider will then report validation errors and warnings encountered
38  * during the unmarshal, marshal, and validate operations to these event
39  * handlers.
40  *
41  * <p>
42  * If the <tt>handleEvent</tt> method throws an unchecked runtime exception,
43  * the JAXB Provider must treat that as if the method returned false, effectively
44  * terminating whatever operation was in progress at the time (unmarshal,
45  * validate, or marshal).
46  *
47  * <p>
48  * Modifying the Java content tree within your event handler is undefined
49  * by the specification and may result in unexpected behaviour.
50  *
51  * <p>
52  * Failing to return false from the <tt>handleEvent</tt> method after
53  * encountering a fatal error is undefined by the specification and may result
54  * in unexpected behavior.
55  *
56  * <p>
57  * <b>Default Event Handler</b>
58  * <blockquote>
59  * See: <a href="Validator.html#defaulthandler">Validator javadocs</a>

```

```

60 * </blockquote>
61 *
62 * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
        Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
63 * @see Unmarshaller
64 * @see Validator
65 * @see Marshaller
66 * @see ValidationEvent
67 * @see javax.xml.bind.util.ValidationEventCollector
68 * @since JAXB1.0
69 */
70 public interface ValidationEventHandler {
71     /**
72      * Receive notification of a validation warning or error.
73      *
74      * The ValidationEvent will have a
75      * {@link ValidationEventLocator ValidationEventLocator} embedded in it that
76      * indicates where the error or warning occurred.
77      *
78      * <p>
79      * If an unchecked runtime exception is thrown from this method, the JAXB
80      * provider will treat it as if the method returned false and interrupt
81      * the current unmarshal, validate, or marshal operation.
82      *
83      * @param event the encapsulated validation event information. It is a
84      * provider error if this parameter is null.
85      * @return true if the JAXB Provider should attempt to continue the current
86      * unmarshal, validate, or marshal operation after handling this
87      * warning/error, false if the provider should terminate the current
88      * operation with the appropriate <tt>UnmarshallerException</tt>,
89      * <tt>ValidationException</tt>, or <tt>MarshalException</tt>.
90      * @throws IllegalArgumentException if the event object is null.
91      */
92     public boolean handleEvent( ValidationEvent event );
93
94 }

```

## 2.29 ValidationEventLocator.java

Secuencia 31: javax/xml/bind/ValidationEventLocator.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind;
27
28 /**
29  * Encapsulate the location of a ValidationEvent.
30  *
31  * <p>
32  * The <tt>ValidationEventLocator</tt> indicates where the <tt>ValidationEvent
33  * </tt> occurred. Different fields will be set depending on the type of
34  * validation that was being performed when the error or warning was detected.
35  * For example, on-demand validation would produce locators that contained
36  * references to objects in the Java content tree while unmarshal-time
37  * validation would produce locators containing information appropriate to the
38  * source of the XML data (file, url, Node, etc).
39  *
40  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
41  * Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
42  * @see Validator
43  * @see ValidationEvent
44  * @since JAXB1.0
45  */
46 public interface ValidationEventLocator {
47     /**
48      * Return the name of the XML source as a URL if available
49      *
50      * @return the name of the XML source as a URL or null if unavailable
51      */
52     public java.net.URL getURL();
53
54     /**
55      * Return the byte offset if available
56      *
57      * @return the byte offset into the input source or -1 if unavailable
58      */
59     public int getOffset();
60
61     /**
62      * Return the line number if available
63      *
64      * @return the line number or -1 if unavailable
65      */
66 }
```

```

66     public int getLineNumber();
67
68     /**
69      * Return the column number if available
70      *
71      * @return the column number or -1 if unavailable
72      */
73     public int getColumnNumber();
74
75     /**
76      * Return a reference to the object in the Java content tree if available
77      *
78      * @return a reference to the object in the Java content tree or null if
79      *         unavailable
80      */
81     public java.lang.Object getObject();
82
83     /**
84      * Return a reference to the DOM Node if available
85      *
86      * @return a reference to the DOM Node or null if unavailable
87      */
88     public org.w3c.dom.Node getNode();
89
90 }

```

## 2.30 ValidationException.java

Secuencia 32: javax/xml/bind/ValidationException.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */

```



```

25
26 package javax.xml.bind;
27
28 /**
29  * This exception indicates that an error has occurred while performing
30  * a validate operation.
31  *
32  * <p>
33  * The <tt>ValidationEventHandler</tt> can cause this exception to be thrown
34  * during the validate operations. See
35  * {@link ValidationEventHandler#handleEvent(ValidationEvent)}
36  * ValidationEventHandler.handleEvent(ValidationEvent)}.
37  *
38  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
39  * @see JAXBException
40  * @see Validator
41  * @since JAXB1.0
42  */
43 public class ValidationException extends JAXBException {
44
45     /**
46      * Construct an ValidationException with the specified detail message. The
47      * errorCode and linkedException will default to null.
48      *
49      * @param message a description of the exception
50      */
51     public ValidationException(String message) {
52         this( message, null, null );
53     }
54
55     /**
56      * Construct an ValidationException with the specified detail message and vendor
57      * specific errorCode. The linkedException will default to null.
58      *
59      * @param message a description of the exception
60      * @param errorCode a string specifying the vendor specific error code
61      */
62     public ValidationException(String message, String errorCode) {
63         this( message, errorCode, null );
64     }
65
66     /**
67      * Construct an ValidationException with a linkedException. The detail message and
68      * vendor specific errorCode will default to null.
69      *
70      * @param exception the linked exception
71      */
72     public ValidationException(Throwable exception) {
73         this( null, null, exception );
74     }
75
76     /**
77      * Construct an ValidationException with the specified detail message and

```

```

78     * linkedException. The errorCode will default to null.
79     *
80     * @param message a description of the exception
81     * @param exception the linked exception
82     */
83     public ValidationException(String message, Throwable exception) {
84         this( message, null, exception );
85     }
86
87     /**
88     * Construct an ValidationException with the specified detail message, vendor
89     * specific errorCode, and linkedException.
90     *
91     * @param message a description of the exception
92     * @param errorCode a string specifying the vendor specific error code
93     * @param exception the linked exception
94     */
95     public ValidationException(String message, String errorCode, Throwable exception) {
96         super( message, errorCode, exception );
97     }
98
99 }

```

## 2.31 Validator.java

Secuencia 33: javax/xml/bind/Validator.java

```

1  /*
2   * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind;
27

```

```

28 /**
29  * As of JAXB 2.0, this class is deprecated and optional.
30  * <p>
31  * The <tt>Validator</tt> class is responsible for controlling the validation
32  * of content trees during runtime.
33  *
34  * <p>
35  * <a name="validationtypes"></a>
36  * <b>Three Forms of Validation</b><br>
37  * <blockquote>
38  *   <dl>
39  *     <dt><b>Unmarshal-Time Validation</b></dt>
40  *     <dd>This form of validation enables a client application to receive
41  *       information about validation errors and warnings detected while
42  *       unmarshalling XML data into a Java content tree and is completely
43  *       orthogonal to the other types of validation. To enable or disable
44  *       it, see the javadoc for
45  *       {@link Unmarshaller#setValidating(boolean) Unmarshaller.setValidating}.
46  *       All JAXB 1.0 Providers are required to support this operation.
47  *     </dd>
48  *
49  *     <dt><b>On-Demand Validation</b></dt>
50  *     <dd> This form of validation enables a client application to receive
51  *       information about validation errors and warnings detected in the
52  *       Java content tree. At any point, client applications can call
53  *       the {@link Validator#validate(Object) Validator.validate} method
54  *       on the Java content tree (or any sub-tree of it). All JAXB 1.0
55  *       Providers are required to support this operation.
56  *     </dd>
57  *
58  *     <dt><b>Fail-Fast Validation</b></dt>
59  *     <dd> This form of validation enables a client application to receive
60  *       immediate feedback about modifications to the Java content tree
61  *       that violate type constraints on Java Properties as defined in
62  *       the specification. JAXB Providers are not required support
63  *       this type of validation. Of the JAXB Providers that do support
64  *       this type of validation, some may require you to decide at schema
65  *       compile time whether or not a client application will be allowed
66  *       to request fail-fast validation at runtime.
67  *     </dd>
68  *   </dl>
69  * </blockquote>
70  *
71  * <p>
72  * The <tt>Validator</tt> class is responsible for managing On-Demand Validation.
73  * The <tt>Unmarshaller</tt> class is responsible for managing Unmarshal-Time
74  * Validation during the unmarshal operations. Although there is no formal
75  * method of enabling validation during the marshal operations, the
76  * <tt>Marshaller</tt> may detect errors, which will be reported to the
77  * <tt>ValidationEventHandler</tt> registered on it.
78  *
79  * <p>
80  * <a name="defaultHandler"></a>

```

```

81 * <b>Using the Default EventHandler</b><br>
82 * <blockquote>
83 *   If the client application does not set an event handler on their
84 *   <tt>Validator</tt>, <tt>Unmarshaller</tt>, or <tt>Marshaller</tt> prior to
85 *   calling the validate, unmarshal, or marshal methods, then a default event
86 *   handler will receive notification of any errors or warnings encountered.
87 *   The default event handler will cause the current operation to halt after
88 *   encountering the first error or fatal error (but will attempt to continue
89 *   after receiving warnings).
90 * </blockquote>
91 *
92 * <p>
93 * <a name="handlingevents"></a>
94 * <b>Handling Validation Events</b><br>
95 * <blockquote>
96 *   There are three ways to handle events encountered during the unmarshal,
97 *   validate, and marshal operations:
98 *   <dl>
99 *     <dt>Use the default event handler</dt>
100 *     <dd>The default event handler will be used if you do not specify one
101 *       via the <tt>setEventHandler</tt> API's on <tt>Validator</tt>,
102 *       <tt>Unmarshaller</tt>, or <tt>Marshaller</tt>.
103 *     </dd>
104 *
105 *     <dt>Implement and register a custom event handler</dt>
106 *     <dd>Client applications that require sophisticated event processing
107 *       can implement the <tt>ValidationEventHandler</tt> interface and
108 *       register it with the <tt>Unmarshaller</tt> and/or
109 *       <tt>Validator</tt>.
110 *     </dd>
111 *
112 *     <dt>Use the {@link javax.xml.bind.util.ValidationEventCollector ValidationEventCollector}
113 *       utility</dt>
114 *     <dd>For convenience, a specialized event handler is provided that
115 *       simply collects any <tt>ValidationEvent</tt> objects created
116 *       during the unmarshal, validate, and marshal operations and
117 *       returns them to the client application as a
118 *       <tt>java.util.Collection</tt>.
119 *     </dd>
120 *   </dl>
121 * </blockquote>
122 *
123 * <p>
124 * <b>Validation and Well-Formedness</b><br>
125 * <blockquote>
126 * <p>
127 * Validation events are handled differently depending on how the client
128 * application is configured to process them as described in the previous
129 * section. However, there are certain cases where a JAXB Provider indicates
130 * that it is no longer able to reliably detect and report errors. In these
131 * cases, the JAXB Provider will set the severity of the ValidationEvent to
132 * FATAL_ERROR to indicate that the unmarshal, validate, or marshal operations
133 * should be terminated. The default event handler and

```

```

134 * <tt>ValidationEventCollector</tt> utility class must terminate processing
135 * after being notified of a fatal error. Client applications that supply their
136 * own <tt>ValidationEventHandler</tt> should also terminate processing after
137 * being notified of a fatal error. If not, unexpected behaviour may occur.
138 * </blockquote>
139 *
140 * <p>
141 * <a name="supportedProps"></a>
142 * <b>Supported Properties</b><br>
143 * </blockquote>
144 * <p>
145 * There currently are not any properties required to be supported by all
146 * JAXB Providers on Validator. However, some providers may support
147 * their own set of provider specific properties.
148 * </blockquote>
149 *
150 *
151 * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li><li>Kohsuke Kawaguchi, Sun
152 *   Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
153 * @see JAXBContext
154 * @see Unmarshaller
155 * @see ValidationEventHandler
156 * @see ValidationEvent
157 * @see javax.xml.bind.util.ValidationEventCollector
158 * @since JAXB1.0
159 * @deprecated since JAXB 2.0
160 */
161 public interface Validator {
162     /**
163      * Allow an application to register a validation event handler.
164      * <p>
165      * The validation event handler will be called by the JAXB Provider if any
166      * validation errors are encountered during calls to
167      * {@link #validate(Object) validate}. If the client application does not
168      * register a validation event handler before invoking the validate method,
169      * then validation events will be handled by the default event handler which
170      * will terminate the validate operation after the first error or fatal error
171      * is encountered.
172      * <p>
173      * Calling this method with a null parameter will cause the Validator
174      * to revert back to the default default event handler.
175      *
176      * @param handler the validation event handler
177      * @throws JAXBException if an error was encountered while setting the
178      *   event handler
179      * @deprecated since JAXB2.0
180      */
181     public void setEventHandler( ValidationEventHandler handler )
182         throws JAXBException;
183
184     /**
185      * Return the current event handler or the default event handler if one

```

```

186     * hasn't been set.
187     *
188     * @return the current ValidationEventHandler or the default event handler
189     *         if it hasn't been set
190     * @throws JAXBException if an error was encountered while getting the
191     *         current event handler
192     * @deprecated since JAXB2.0
193     */
194     public ValidationEventHandler getEventHandler()
195         throws JAXBException;
196
197     /**
198     * Validate the Java content tree starting at <tt>subrootObj</tt>.
199     * <p>
200     * Client applications can use this method to validate Java content trees
201     * on-demand at runtime. This method can be used to validate any arbitrary
202     * subtree of the Java content tree. Global constraint checking <b>will not
203     * </b> be performed as part of this operation (i.e. ID/IDREF constraints).
204     *
205     * @param subrootObj the obj to begin validation at
206     * @throws JAXBException if any unexpected problem occurs during validation
207     * @throws ValidationException
208     *         If the {@link ValidationEventHandler ValidationEventHandler}
209     *         returns false from its <tt>handleEvent</tt> method or the
210     *         <tt>Validator</tt> is unable to validate the content tree rooted
211     *         at <tt>subrootObj</tt>
212     * @throws IllegalArgumentException
213     *         If the subrootObj parameter is null
214     * @return true if the subtree rooted at <tt>subrootObj</tt> is valid, false
215     *         otherwise
216     * @deprecated since JAXB2.0
217     */
218     public boolean validate( Object subrootObj ) throws JAXBException;
219
220     /**
221     * Validate the Java content tree rooted at <tt>rootObj</tt>.
222     * <p>
223     * Client applications can use this method to validate Java content trees
224     * on-demand at runtime. This method is used to validate an entire Java
225     * content tree. Global constraint checking <b>will</b> be performed as
226     * part of this operation (i.e. ID/IDREF constraints).
227     *
228     * @param rootObj the root obj to begin validation at
229     * @throws JAXBException if any unexpected problem occurs during validation
230     * @throws ValidationException
231     *         If the {@link ValidationEventHandler ValidationEventHandler}
232     *         returns false from its <tt>handleEvent</tt> method or the
233     *         <tt>Validator</tt> is unable to validate the content tree rooted
234     *         at <tt>rootObj</tt>
235     * @throws IllegalArgumentException
236     *         If the rootObj parameter is null
237     * @return true if the tree rooted at <tt>rootObj</tt> is valid, false
238     *         otherwise

```

```

239     * @deprecated since JAXB2.0
240     */
241     public boolean validateRoot( Object rootObj ) throws JAXBException;
242
243     /**
244     * Set the particular property in the underlying implementation of
245     * <tt>Validator</tt>. This method can only be used to set one of
246     * the standard JAXB defined properties above or a provider specific
247     * property. Attempting to set an undefined property will result in
248     * a PropertyException being thrown. See <a href="#supportedProps">
249     * Supported Properties</a>.
250     *
251     * @param name the name of the property to be set. This value can either
252     *             be specified using one of the constant fields or a user
253     *             supplied string.
254     * @param value the value of the property to be set
255     *
256     * @throws PropertyException when there is an error processing the given
257     *             property or value
258     * @throws IllegalArgumentException
259     *             If the name parameter is null
260     * @deprecated since JAXB2.0
261     */
262     public void setProperty( String name, Object value )
263         throws PropertyException;
264
265     /**
266     * Get the particular property in the underlying implementation of
267     * <tt>Validator</tt>. This method can only be used to get one of
268     * the standard JAXB defined properties above or a provider specific
269     * property. Attempting to get an undefined property will result in
270     * a PropertyException being thrown. See <a href="#supportedProps">
271     * Supported Properties</a>.
272     *
273     * @param name the name of the property to retrieve
274     * @return the value of the requested property
275     *
276     * @throws PropertyException
277     *             when there is an error retrieving the given property or value
278     *             property name
279     * @throws IllegalArgumentException
280     *             If the name parameter is null
281     * @deprecated since JAXB2.0
282     */
283     public Object getProperty( String name ) throws PropertyException;
284
285 }

```

## 2.32 WhiteSpaceProcessor.java

Secuencia 34: javax/xml/bind/WhiteSpaceProcessor.java

```

1  /*
2  * Copyright (c) 2007, 2013, Oracle and/or its affiliates. All rights reserved.

```

```
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind;
27
28 /**
29  * Processes white space normalization.
30  *
31  * @since 1.0
32  */
33 abstract class WhiteSpaceProcessor {
34
35  // benchmarking (see test/src/ReplaceTest.java in the CVS Attic)
36  // showed that this code is slower than the current code.
37  //
38  //     public static String replace(String text) {
39  //         final int len = text.length();
40  //         StringBuffer result = new StringBuffer(len);
41  //
42  //         for (int i = 0; i < len; i++) {
43  //             char ch = text.charAt(i);
44  //             if (isWhiteSpace(ch))
45  //                 result.append(' ');
46  //             else
47  //                 result.append(ch);
48  //         }
49  //
50  //         return result.toString();
51  //     }
52
53     public static String replace(String text) {
54         return replace( (CharSequence)text ).toString();
55     }
```



```
56
57 /**
58  * @since 2.0
59  */
60 public static CharSequence replace(CharSequence text) {
61     int i=text.length()-1;
62
63     // look for the first whitespace char.
64     while( i>=0 && !isWhiteSpaceExceptSpace(text.charAt(i)) )
65         i--;
66
67     if( i<0 )
68         // no such whitespace. replace(text)==text.
69         return text;
70
71     // we now know that we need to modify the text.
72     // allocate a char array to do it.
73     StringBuilder buf = new StringBuilder(text);
74
75     buf.setCharAt(i--, ' ');
76     for( ; i>=0; i-- )
77         if( isWhiteSpaceExceptSpace(buf.charAt(i))
78             buf.setCharAt(i, ' ');
79
80     return new String(buf);
81 }
82
83 /**
84  * Equivalent of {@link String#trim()}.
85  * @since 2.0
86  */
87 public static CharSequence trim(CharSequence text) {
88     int len = text.length();
89     int start = 0;
90
91     while( start<len && isWhiteSpace(text.charAt(start)) )
92         start++;
93
94     int end = len-1;
95
96     while( end>start && isWhiteSpace(text.charAt(end)) )
97         end--;
98
99     if(start==0 && end==len-1)
100         return text;    // no change
101     else
102         return text.subSequence(start,end+1);
103 }
104
105 public static String collapse(String text) {
106     return collapse( (CharSequence)text ).toString();
107 }
108
```

```

109  /**
110   * This is usually the biggest processing bottleneck.
111   *
112   * @since 2.0
113   */
114  public static CharSequence collapse(CharSequence text) {
115      int len = text.length();
116
117      // most of the texts are already in the collapsed form.
118      // so look for the first whitespace in the hope that we will
119      // never see it.
120      int s=0;
121      while(s<len) {
122          if(isWhiteSpace(text.charAt(s)))
123              break;
124          s++;
125      }
126      if(s==len)
127          // the input happens to be already collapsed.
128          return text;
129
130      // we now know that the input contains spaces.
131      // let's sit down and do the collapsing normally.
132
133      StringBuilder result = new StringBuilder(len /*allocate enough size to avoid re-allocation
134          */ );
135
136      if(s!=0) {
137          for( int i=0; i<s; i++ )
138              result.append(text.charAt(i));
139          result.append(' ');
140      }
141
142      boolean inStripMode = true;
143      for (int i = s+1; i < len; i++) {
144          char ch = text.charAt(i);
145          boolean b = isWhiteSpace(ch);
146          if (inStripMode && b)
147              continue; // skip this character
148
149          inStripMode = b;
150          if (inStripMode)
151              result.append(' ');
152          else
153              result.append(ch);
154      }
155
156      // remove trailing whitespaces
157      len = result.length();
158      if (len > 0 && result.charAt(len - 1) == ' ')
159          result.setLength(len - 1);
160      // whitespaces are already collapsed,
161      // so all we have to do is to remove the last one character

```

```

161     // if it's a whitespace.
162
163     return result;
164 }
165
166 /**
167  * Returns true if the specified string is all whitespace.
168  */
169 public static final boolean isWhiteSpace(CharSequence s) {
170     for( int i=s.length()-1; i>=0; i-- )
171         if(!isWhiteSpace(s.charAt(i)))
172             return false;
173     return true;
174 }
175
176 /** returns true if the specified char is a white space character. */
177 public static final boolean isWhiteSpace(char ch) {
178     // most of the characters are non-control characters.
179     // so check that first to quickly return false for most of the cases.
180     if( ch>0x20 ) return false;
181
182     // other than we have to do four comparisons.
183     return ch == 0x9 || ch == 0xA || ch == 0xD || ch == 0x20;
184 }
185
186 /**
187  * Returns true if the specified char is a white space character
188  * but not 0x20.
189  */
190 protected static final boolean isWhiteSpaceExceptSpace(char ch) {
191     // most of the characters are non-control characters.
192     // so check that first to quickly return false for most of the cases.
193     if( ch>=0x20 ) return false;
194
195     // other than we have to do four comparisons.
196     return ch == 0x9 || ch == 0xA || ch == 0xD;
197 }
198 }

```

### 3 Xml Bind Annotation (javax.xml.bind.annotation package)

#### 3.1 DomHandler.java

Secuencia 35: javax/xml/bind/annotation/DomHandler.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```

```
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.bind.annotation;
27
28  import javax.xml.bind.ValidationEventHandler;
29  import javax.xml.transform.Result;
30  import javax.xml.transform.Source;
31
32  /**
33   * Converts an element (and its descendants)
34   * from/to DOM (or similar) representation.
35   *
36   * <p>
37   * Implementations of this interface will be used in conjunction with
38   * {@link XmlAnyElement} annotation to map an element of XML into a representation
39   * of infoset such as W3C DOM.
40   *
41   * <p>
42   * Implementations hide how a portion of XML is converted into/from such
43   * DOM-like representation, allowing JAXB providers to work with arbitrary
44   * such library.
45   *
46   * <P>
47   * This interface is intended to be implemented by library writers
48   * and consumed by JAXB providers. None of those methods are intended to
49   * be called from applications.
50   *
51   * @author Kohsuke Kawaguchi
52   * @since JAXB2.0
53   */
54  public interface DomHandler<ElementT,ResultT extends Result> {
55      /**
56       * When a JAXB provider needs to unmarshal a part of a document into an
57       * infoset representation, it first calls this method to create a
58       * {@link Result} object.
59       *
60       * <p>
61       * A JAXB provider will then send a portion of the XML
62       * into the given result. Such a portion always form a subtree
```

```

63      * of the whole XML document rooted at an element.
64      *
65      * @param errorHandler
66      *     if any error happens between the invocation of this method
67      *     and the invocation of {@link #getElement(Result)}, they
68      *     must be reported to this handler.
69      *
70      *     The caller must provide a non-null error handler.
71      *
72      *     The {@link Result} object created from this method
73      *     may hold a reference to this error handler.
74      *
75      * @return
76      *     null if the operation fails. The error must have been reported
77      *     to the error handler.
78      */
79      ResultT createUnmarshaller( ValidationEventHandler errorHandler );
80
81      /**
82      * Once the portion is sent to the {@link Result}. This method is called
83      * by a JAXB provider to obtain the unmarshalled element representation.
84      *
85      * <p>
86      * Multiple invocations of this method may return different objects.
87      * This method can be invoked only when the whole sub-tree are fed
88      * to the {@link Result} object.
89      *
90      * @param rt
91      *     The {@link Result} object created by {@link #createUnmarshaller(ValidationEventHandler)
92      *     }.
93      *
94      * @return
95      *     null if the operation fails. The error must have been reported
96      *     to the error handler.
97      */
98      ElementT getElement(ResultT rt);
99
100     /**
101     * This method is called when a JAXB provider needs to marshal an element
102     * to XML.
103     *
104     * <p>
105     * If non-null, the returned {@link Source} must contain a whole document
106     * rooted at one element, which will then be weaved into a bigger document
107     * that the JAXB provider is marshalling.
108     *
109     * @param errorHandler
110     *     Receives any errors happened during the process of converting
111     *     an element into a {@link Source}.
112     *
113     *     The caller must provide a non-null error handler.
114     *
115     * @return

```

```

115     *      null if there was an error. The error should have been reported
116     *      to the handler.
117     */
118     Source marshal( ElementT n, ValidationEventHandler errorHandler );
119 }

```

### 3.2 W3CDomHandler.java

Secuencia 36: javax.xml.bind.annotation.W3CDomHandler.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import org.w3c.dom.Document;
29 import org.w3c.dom.DocumentFragment;
30 import org.w3c.dom.Element;
31 import org.w3c.dom.Node;
32
33 import javax.xml.bind.ValidationEventHandler;
34 import javax.xml.parsers.DocumentBuilder;
35 import javax.xml.transform.Source;
36 import javax.xml.transform.dom.DOMResult;
37 import javax.xml.transform.dom.DOMSource;
38
39 /**
40 * {@link DomHandler} implementation for W3C DOM (<code>org.w3c.dom</code> package.)
41 *
42 * @author Kohsuke Kawaguchi
43 * @since JAXB2.0
44 */

```

```
45 public class W3CDomHandler implements DomHandler<Element, DOMResult> {
46
47     private DocumentBuilder builder;
48
49     /**
50      * Default constructor.
51      *
52      * It is up to a JAXB provider to decide which DOM implementation
53      * to use or how that is configured.
54      */
55     public W3CDomHandler() {
56         this.builder = null;
57     }
58
59     /**
60      * Constructor that allows applications to specify which DOM implementation
61      * to be used.
62      *
63      * @param builder
64      *     must not be null. JAXB uses this {@link DocumentBuilder} to create
65      *     a new element.
66      */
67     public W3CDomHandler(DocumentBuilder builder) {
68         if(builder==null)
69             throw new IllegalArgumentException();
70         this.builder = builder;
71     }
72
73     public DocumentBuilder getBuilder() {
74         return builder;
75     }
76
77     public void setBuilder(DocumentBuilder builder) {
78         this.builder = builder;
79     }
80
81     public DOMResult createUnmarshaller(ValidationEventHandler errorHandler) {
82         if(builder==null)
83             return new DOMResult();
84         else
85             return new DOMResult(builder.newDocument());
86     }
87
88     public Element getElement(DOMResult r) {
89         // JAXP spec is ambiguous about what really happens in this case,
90         // so work defensively
91         Node n = r.getNode();
92         if( n instanceof Document ) {
93             return ((Document)n).getDocumentElement();
94         }
95         if( n instanceof Element )
96             return (Element)n;
97         if( n instanceof DocumentFragment )
```

```

98         return (Element)n.getChildNodes().item(0);
99
100        // if the result object contains something strange,
101        // it is not a user problem, but it is a JAXB provider's problem.
102        // That's why we throw a runtime exception.
103        throw new IllegalStateException(n.toString());
104    }
105
106    public Source marshal(Element element, ValidationEventHandler errorHandler) {
107        return new DOMSource(element);
108    }
109 }

```

### 3.3 XmlAccessOrder.java

Secuencia 37: javax/xml/bind/annotation/XmlAccessOrder.java

```

1  /*
2   * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 /**
29  * Used by XmlAccessorOrder to control the ordering of properties and
30  * fields in a JAXB bound class.
31  *
32  * @author Sekhar Vajjhala, Sun Microsystems, Inc.
33  * @since JAXB2.0
34  * @see XmlAccessorOrder
35  */
36
37 public enum XmlAccessOrder {

```



```

38  /**
39   * The ordering of fields and properties in a class is undefined.
40   */
41  UNDEFINED,
42  /**
43   * The ordering of fields and properties in a class is in
44   * alphabetical order as determined by the
45   * method java.lang.String.compareTo(String anotherString).
46   */
47  ALPHABETICAL
48  }

```

### 3.4 XmlAccessType.java

Secuencia 38: javax/xml/bind/annotation/XmlAccessType.java

```

1  /*
2   * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28
29
30 /**
31  * Used by XmlAccessorType to control serialization of fields or
32  * properties.
33  *
34  * @author Sekhar Vajjhala, Sun Microsystems, Inc.
35  * @since JAXB2.0
36  * @see XmlAccessorType
37  */
38

```

```

39 public enum XmlAccessType {
40     /**
41      * Every getter/setter pair in a JAXB-bound class will be automatically
42      * bound to XML, unless annotated by {@link XmlTransient}.
43      *
44      * Fields are bound to XML only when they are explicitly annotated
45      * by some of the JAXB annotations.
46      */
47     PROPERTY,
48     /**
49      * Every non static, non transient field in a JAXB-bound class will be automatically
50      * bound to XML, unless annotated by {@link XmlTransient}.
51      *
52      * Getter/setter pairs are bound to XML only when they are explicitly annotated
53      * by some of the JAXB annotations.
54      */
55     FIELD,
56     /**
57      * Every public getter/setter pair and every public field will be
58      * automatically bound to XML, unless annotated by {@link XmlTransient}.
59      *
60      * Fields or getter/setter pairs that are private, protected, or
61      * defaulted to package-only access are bound to XML only when they are
62      * explicitly annotated by the appropriate JAXB annotations.
63      */
64     PUBLIC_MEMBER,
65     /**
66      * None of the fields or properties is bound to XML unless they
67      * are specifically annotated with some of the JAXB annotations.
68      */
69     NONE
70 }

```

### 3.5 XmlAccessorOrder.java

Secuencia 39: javax/xml/bind/annotation/XmlAccessorOrder.java

```

1  /*
2   * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *

```

```

18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.Inherited;
32
33 import static java.lang.annotation.ElementType.*;
34 import static java.lang.annotation.RetentionPolicy.*;
35
36 /**
37  * <p> Controls the ordering of fields and properties in a class. </p>
38  *
39  * <h3>Usage </h3>
40  *
41  * <p> <tt> @XmlAccessorType </tt> annotation can be used with the following
42  * program elements:</p>
43  *
44  * <ul>
45  *   <li> package</li>
46  *   <li> a top level class </li>
47  * </ul>
48  *
49  * <p> See "Package Specification" in <tt>javax.xml.bind</tt> package javadoc for
50  * additional common information.</p>
51  *
52  * <p>The effective {@link XmlAccessorType} on a class is determined
53  * as follows:
54  *
55  * <ul>
56  *   <li> If there is a <tt>@XmlAccessorType</tt> on a class, then
57  *     it is used. </li>
58  *   <li> Otherwise, if a <tt>@XmlAccessorType </tt> exists on one of
59  *     its super classes, then it is inherited (by the virtue of
60  *     {@link Inherited})
61  *   <li> Otherwise, the <tt>@XmlAccessorType</tt> on the package
62  *     of the class is used, if it's there.
63  *   <li> Otherwise {@link XmlAccessorType#UNDEFINED}.
64  * </ul>
65  *
66  * <p>This annotation can be used with the following annotations:
67  *   {@link XmlType}, {@link XmlRootElement}, {@link XmlAccessorType},
68  *   {@link XmlSchema}, {@link XmlSchemaType}, {@link XmlSchemaTypes},
69  *   , {@link XmlJavaTypeAdapter}. It can also be used with the
70  *   following annotations at the package level: {@link XmlJavaTypeAdapter}.

```

```

71  *
72  * @author Sekhar Vajjhala, Sun Microsystems, Inc.
73  * @since JAXB2.0
74  * @see XmlAccessOrder
75  */
76
77 @Inherited @Retention(RUNTIME) @Target({PACKAGE, TYPE})
78 public @interface XmlAccessOrder {
79     XmlAccessOrder value() default XmlAccessOrder.UNDEFINED;
80 }

```

### 3.6 XmlAccessorType.java

Secuencia 40: javax/xml/bind/annotation/XmlAccessorType.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.Inherited;
32
33 import static java.lang.annotation.ElementType.*;
34 import static java.lang.annotation.RetentionPolicy.*;
35
36 /**
37  * <p> Controls whether fields or Javabeen properties are serialized by default. </p>
38  *
39  * <p> <b> Usage </b> </p>

```

```

40  *
41  * <p> <tt>@XmlAccessorType</tt> annotation can be used with the following program elements:</p>
42  *
43  * <ul>
44  *   <li> package</li>
45  *   <li> a top level class </li>
46  * </ul>
47  *
48  * <p> See "Package Specification" in javax.xml.bind.package javadoc for
49  * additional common information.</p>
50  *
51  * <p>This annotation provides control over the default serialization
52  * of properties and fields in a class.
53  *
54  * <p>The annotation <tt> @XmlAccessorType </tt> on a package applies to
55  * all classes in the package. The following inheritance
56  * semantics apply:
57  *
58  * <ul>
59  *   <li> If there is a <tt>@XmlAccessorType</tt> on a class, then it
60  *     is used. </li>
61  *   <li> Otherwise, if a <tt>@XmlAccessorType</tt> exists on one of
62  *     its super classes, then it is inherited.
63  *   <li> Otherwise, the <tt>@XmlAccessorType </tt> on a package is
64  *     inherited.
65  * </ul>
66  * <p> <b> Defaulting Rules: </b> </p>
67  *
68  * <p>By default, if <tt>@XmlAccessorType</tt> on a package is absent,
69  * then the following package level annotation is assumed.</p>
70  * <pre>
71  *   &#64;XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
72  * </pre>
73  * <p> By default, if <tt>@XmlAccessorType</tt> on a class is absent,
74  * and none of its super classes is annotated with
75  * <tt>@XmlAccessorType</tt>, then the following default on the class
76  * is assumed: </p>
77  * <pre>
78  *   &#64;XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
79  * </pre>
80  * <p>This annotation can be used with the following annotations:
81  *   {@link XmlType}, {@link XmlRootElement}, {@link XmlAccessorType},
82  *   {@link XmlSchema}, {@link XmlSchemaType}, {@link XmlSchemaTypes},
83  *   , {@link XmlJavaTypeAdapter}. It can also be used with the
84  *   following annotations at the package level: {@link XmlJavaTypeAdapter}.
85  *
86  * @author Sekhar Vajjhala, Sun Microsystems, Inc.
87  * @since JAXB2.0
88  * @see XmlAccessType
89  */
90
91 @Inherited @Retention(RUNTIME) @Target({PACKAGE, TYPE})
92 public @interface XmlAccessorType {

```

```

93
94  /**
95   * Specifies whether fields or properties are serialized.
96   *
97   * @see XmlAccessType
98   */
99   XmlAccessType value() default XmlAccessType.PUBLIC_MEMBER;
100 }

```

### 3.7 XmlAnyAttribute.java

Secuencia 41: javax/xml/bind/annotation/XmlAnyAttribute.java

```

1  /*
2   * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.namespace.QName;
29 import java.lang.annotation.Retention;
30 import java.lang.annotation.Target;
31 import java.util.Map;
32
33 import static java.lang.annotation.RetentionPolicy.RUNTIME;
34 import static java.lang.annotation.ElementType.FIELD;
35 import static java.lang.annotation.ElementType.METHOD;
36
37 /**
38  * <p>
39  * Maps a JavaBean property to a map of wildcard attributes.
40  *
41  * <p> <b>Usage</b> </p>

```

```

42 * <p>
43 * The <tt>#64;XmlAnyAttribute</tt> annotation can be used with the
44 * following program elements:
45 * <ul>
46 *   <li> JavaBean property </li>
47 *   <li> non static, non transient field </li>
48 * </ul>
49 *
50 * <p>See "Package Specification" in javax.xml.bind.package javadoc for
51 * additional common information.</p>
52 *
53 * The usage is subject to the following constraints:
54 * <ul>
55 *   <li> At most one field or property in a class can be annotated
56 *     with <tt>#64;XmlAnyAttribute</tt>. </li>
57 *   <li> The type of the property or the field must <tt>java.util.Map</tt> </li>
58 * </ul>
59 *
60 * <p>
61 * While processing attributes to be unmarshalled into a value class,
62 * each attribute that is not statically associated with another
63 * JavaBean property, via {@link XmlAttribute}, is entered into the
64 * wildcard attribute map represented by
65 * {@link Map}&lt;{@link QName},{@link Object}>. The attribute QName is the
66 * map's key. The key's value is the String value of the attribute.
67 *
68 * @author Kohsuke Kawaguchi, Sun Microsystems, Inc.
69 * @since JAXB2.0
70 */
71 @Retention(RUNTIME)
72 @Target({FIELD,METHOD})
73 public @interface XmlAnyAttribute {
74 }

```

### 3.8 XmlAnyElement.java

Secuencia 42: javax/xml/bind/annotation/XmlAnyElement.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *

```

```
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import org.w3c.dom.Element;
29
30 import javax.xml.bind.JAXBContext;
31 import javax.xml.bind.JAXBElement;
32 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
33 import java.lang.annotation.Retention;
34 import java.lang.annotation.Target;
35 import java.util.List;
36
37 import static java.lang.annotation.ElementType.FIELD;
38 import static java.lang.annotation.ElementType.METHOD;
39 import static java.lang.annotation.RetentionPolicy.RUNTIME;
40
41 /**
42  * Maps a JavaBean property to XML infoset representation and/or JAXB element.
43  *
44  * <p>
45  * This annotation serves as a "catch-all" property while unmarshalling
46  * xml content into a instance of a JAXB annotated class. It typically
47  * annotates a multi-valued JavaBean property, but it can occur on
48  * single value JavaBean property. During unmarshalling, each xml element
49  * that does not match a static &#64;XmlElement or &#64;XmlElementRef
50  * annotation for the other JavaBean properties on the class, is added to this
51  * "catch-all" property.
52  *
53  * <p>
54  * <h2>Usages:</h2>
55  * <pre>
56  * &#64;XmlAnyElement
57  * public {@link Element}[] others;
58  *
59  * // Collection of {@link Element} or JAXB elements.
60  * &#64;XmlAnyElement(lax="true")
61  * public {@link Object}[] others;
62  *
63  * &#64;XmlAnyElement
64  * private List<{@link Element}> nodes;
65  *
66  * &#64;XmlAnyElement
67  * private {@link Element} node;
68  * </pre>
69  *
```



```

70 * <h2>Restriction usage constraints</h2>
71 * <p>
72 * This annotation is mutually exclusive with
73 * {@link XmlElement}, {@link XmlAttribute}, {@link XmlValue},
74 * {@link XmlElements}, {@link XmlID}, and {@link XmlIDREF}.
75 *
76 * <p>
77 * There can be only one {@link XmlAnyElement} annotated JavaBean property
78 * in a class and its super classes.
79 *
80 * <h2>Relationship to other annotations</h2>
81 * <p>
82 * This annotation can be used with {@link XmlJavaTypeAdapter}, so that users
83 * can map their own data structure to DOM, which in turn can be composed
84 * into XML.
85 *
86 * <p>
87 * This annotation can be used with {@link XmlMixed} like this:
88 * <pre>
89 * // List of java.lang.String or DOM nodes.
90 * &#64;XmlAnyElement &#64;XmlMixed
91 * List<Object> others;
92 * </pre>
93 *
94 *
95 * <h2>Schema To Java example</h2>
96 *
97 * The following schema would produce the following Java class:
98 * <pre>
99 * <?xml:namespace="http://www.w3.org/2001/XMLSchema-instance" prefix="xs" />
100 * <xs:sequence>
101 *   <xs:element name="a" type="xs:int" />
102 *   <xs:element name="b" type="xs:int" />
103 *   <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
104 * </xs:sequence>
105 * </xs:complexType>
106 * </pre>
107 *
108 * <pre>
109 * class Foo {
110 *   int a;
111 *   int b;
112 *   &#64;{@link XmlAnyElement}
113 *   List<Element> any;
114 * }
115 * </pre>
116 *
117 * It can unmarshal instances like
118 *
119 * <pre>
120 * <?xml:namespace="http://www.w3.org/2001/XMLSchema-instance" prefix="e" />
121 * <a>1</a>
122 * <e:other /> // this will be bound to DOM, because unmarshalling is orderless

```

```

123 * <b>3</b>
124 * <e:other />
125 * <c>5</c> // this will be bound to DOM, because the annotation doesn't remember
    namespaces.
126 * </foo>
127 * </pre>
128 *
129 *
130 *
131 * The following schema would produce the following Java class:
132 * <pre>
133 * <xs:complexType name="bar">
134 *   <xs:complexContent>
135 *     <xs:extension base="foo">
136 *       <xs:sequence>
137 *         <xs:element name="c" type="xs:int" />
138 *         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"
    />
139 *       </xs:sequence>
140 *     </xs:extension>
141 *   </xs:complexType>
142 * </pre>
143 *
144 * <pre>
145 * class Bar extends Foo {
146 *   int c;
147 *   // Foo.getAny() also represents wildcard content for type definition bar.
148 * }
149 * </pre>
150 *
151 *
152 * It can unmarshal instances like
153 *
154 * <pre>
155 * <bar xmlns:e="extra">
156 *   <a>1</a>
157 *   <e:other /> // this will be bound to DOM, because unmarshalling is orderless
158 *   <b>3</b>
159 *   <e:other />
160 *   <c>5</c> // this now goes to Bar.c
161 *   <e:other /> // this will go to Foo.any
162 * </bar>
163 * </pre>
164 *
165 *
166 *
167 *
168 * <h2>Using {@link XmlAnyElement} with {@link XmlElementRef}</h2>
169 * <p>
170 * The {@link XmlAnyElement} annotation can be used with {@link XmlElementRef}s to
171 * designate additional elements that can participate in the content tree.
172 *
173 * <p>

```

```

174 * The following schema would produce the following Java class:
175 * <pre>
176 * <?xml:complexType name="foo">
177 *   <?xml:choice maxOccurs="unbounded" minOccurs="0">
178 *     <?xml:element name="a" type="xs:int" />
179 *     <?xml:element name="b" type="xs:int" />
180 *     <?xml:any namespace="##other" processContents="lax" />
181 *   </?xml:choice>
182 * </?xml:complexType>
183 * </pre>
184 *
185 * <pre>
186 * class Foo {
187 *   &#64;{@link XmlAnyElement}(lax="true")
188 *   &#64;{@link XmlElementRefs}({
189 *     &#64;{@link XmlElementRef}(name="a", type="JAXBElement.class")
190 *     &#64;{@link XmlElementRef}(name="b", type="JAXBElement.class")
191 *   })
192 *   {@link List}&lt;{@link Object}> others;
193 * }
194 *
195 * &#64;XmlRegistry
196 * class ObjectFactory {
197 *   ...
198 *   &#64;XmlElementDecl(name = "a", namespace = "", scope = Foo.class)
199 *   {@link JAXBElement}&lt;Integer> createFooA( Integer i ) { ... }
200 *
201 *   &#64;XmlElementDecl(name = "b", namespace = "", scope = Foo.class)
202 *   {@link JAXBElement}&lt;Integer> createFooB( Integer i ) { ... }
203 * </pre>
204 *
205 * It can unmarshal instances like
206 *
207 * <pre>
208 * <?xml:foo xmlns:e="extra">
209 *   <?xml:a>1</a>      // this will unmarshal to a {@link JAXBElement} instance whose value is 1.
210 *   <?xml:e:other />    // this will unmarshal to a DOM {@link Element}.
211 *   <?xml:b>3</b>      // this will unmarshal to a {@link JAXBElement} instance whose value is 1.
212 * </?xml:foo>
213 * </pre>
214 *
215 *
216 *
217 *
218 * <h2>W3C XML Schema "lax" wildcard emulation</h2>
219 * The lax element of the annotation enables the emulation of the "lax" wildcard semantics.
220 * For example, when the Java source code is annotated like this:
221 * <pre>
222 * &#64;{@link XmlRootElement}
223 * class Foo {
224 *   &#64;XmlAnyElement(lax=true)
225 *   public {@link Object}[] others;
226 * }

```

```

227 * </pre>
228 * then the following document will unmarshal like this:
229 * <pre>
230 * <foo>
231 *   <unknown />
232 *   <foo />
233 * </foo>
234 *
235 * Foo foo = unmarshal();
236 * // 1 for 'unknown', another for 'foo'
237 * assert foo.others.length==2;
238 * // 'unknown' unmarshals to a DOM element
239 * assert foo.others[0] instanceof Element;
240 * // because of lax=true, the 'foo' element eagerly
241 * // unmarshals to a Foo object.
242 * assert foo.others[1] instanceof Foo;
243 * </pre>
244 *
245 * @author Kohsuke Kawaguchi
246 * @since JAXB2.0
247 */
248 @Retention(RUNTIME)
249 @Target({FIELD,METHOD})
250 public @interface XmlAnyElement {
251
252     /**
253      * Controls the unmarshaller behavior when it sees elements
254      * known to the current {@link JAXBContext}.
255      *
256      * <h3>When false</h3>
257      * <p>
258      * If false, all the elements that match the property will be unmarshalled
259      * to DOM, and the property will only contain DOM elements.
260      *
261      * <h3>When true</h3>
262      * <p>
263      * If true, when an element matches a property marked with {@link XmlAnyElement}
264      * is known to {@link JAXBContext} (for example, there's a class with
265      * {@link XmlRootElement} that has the same tag name, or there's
266      * {@link XmlElementDecl} that has the same tag name),
267      * the unmarshaller will eagerly unmarshal this element to the JAXB object,
268      * instead of unmarshalling it to DOM. Additionally, if the element is
269      * unknown but it has a known xsi:type, the unmarshaller eagerly unmarshals
270      * the element to a {@link JAXBElement}, with the unknown element name and
271      * the JAXBElement value is set to an instance of the JAXB mapping of the
272      * known xsi:type.
273      *
274      * <p>
275      * As a result, after the unmarshalling, the property can become heterogeneous;
276      * it can have both DOM nodes and some JAXB objects at the same time.
277      *
278      * <p>
279      * This can be used to emulate the "lax" wildcard semantics of the W3C XML Schema.

```

```

280     */
281     boolean lax() default false;
282
283     /**
284      * Specifies the {@link DomHandler} which is responsible for actually
285      * converting XML from/to a DOM-like data structure.
286      */
287     Class<? extends DomHandler> value() default W3CDomHandler.class;
288 }

```

### 3.9 XmlAttachmentRef.java

Secuencia 43: javax/xml/bind/annotation/XmlAttachmentRef.java

```

1  /*
2   * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import javax.activation.DataHandler;
29 import static java.lang.annotation.ElementType.*;
30 import java.lang.annotation.Retention;
31 import static java.lang.annotation.RetentionPolicy.RUNTIME;
32 import java.lang.annotation.Target;
33
34 /**
35  * Marks a field/property that its XML form is a uri reference to mime content.
36  * The mime content is optimally stored out-of-line as an attachment.
37  *
38  * A field/property must always map to the {@link DataHandler} class.
39  *
40  * <h2>Usage</h2>

```

```

41 * <pre>
42 * &#64;{@link XmlRootElement}
43 * class Foo {
44 *     &#64;{@link XmlAttachmentRef}
45 *     &#64;{@link XmlAttribute}
46 *     {@link DataHandler} data;
47 *
48 *     &#64;{@link XmlAttachmentRef}
49 *     &#64;{@link XmlElement}
50 *     {@link DataHandler} body;
51 * }
52 * </pre>
53 * The above code maps to the following XML:
54 * <pre>
55 * &lt;xs:element name="foo" xmlns:ref="http://ws-i.org/profiles/basic/1.1/xsd">
56 *     &lt;xs:complexType>
57 *         &lt;xs:sequence>
58 *             &lt;xs:element name="body" type="ref:swaRef" minOccurs="0" />
59 *         &lt;/xs:sequence>
60 *         &lt;xs:attribute name="data" type="ref:swaRef" use="optional" />
61 *     &lt;/xs:complexType>
62 * &lt;/xs:element>
63 * </pre>
64 *
65 * <p>
66 * The above binding supports WS-I AP 1.0 <a href="http://www.ws-i.org/Profiles/AttachmentsProfile
        -1.0-2004-08-24.html#Referencing_Attachments_from_the_SOAP_Envelope">WS-I Attachments Profile
        Version 1.0.</a>
67 *
68 * @author Kohsuke Kawaguchi
69 * @since JAXB2.0
70 */
71 @Retention(RUNTIME)
72 @Target({FIELD,METHOD,PARAMETER})
73 public @interface XmlAttachmentRef {
74 }

```

### 3.10 XmlAttribute.java

Secuencia 44: javax/xml/bind/annotation/XmlAttribute.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```

14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.bind.annotation;
27
28  import java.lang.annotation.Retention;
29  import java.lang.annotation.Target;
30
31  import static java.lang.annotation.ElementType.*;
32  import static java.lang.annotation.RetentionPolicy.*;
33
34  /**
35   * <p>
36   * Maps a JavaBean property to a XML attribute.
37   *
38   * <p> <b>Usage</b> </p>
39   * <p>
40   * The <tt>@XmlAttribute</tt> annotation can be used with the
41   * following program elements:
42   * <ul>
43   *   <li> JavaBean property </li>
44   *   <li> field </li>
45   * </ul>
46   *
47   * <p> A static final field is mapped to a XML fixed attribute.
48   *
49   * <p>See "Package Specification" in javax.xml.bind.package javadoc for
50   * additional common information.</p>
51   *
52   * The usage is subject to the following constraints:
53   * <ul>
54   *   <li> If type of the field or the property is a collection
55   *       type, then the collection item type must be mapped to schema
56   *       simple type.
57   * <pre>
58   *     // Examples
59   *     &#64;XmlAttribute List<Integer> items; //legal
60   *     &#64;XmlAttribute List<Bar> foo; // illegal if Bar does not map to a schema simple type
61   * </pre>
62   *   </li>
63   *   <li> If the type of the field or the property is a non
64   *       collection type, then the type of the property or field
65   *       must map to a simple schema type.
66   * <pre>

```

```

67 * // Examples
68 *      &#64;XmlAttribute int foo; // legal
69 *      &#64;XmlAttribute Foo foo; // illegal if Foo does not map to a schema simple type
70 * </pre>
71 * </li>
72 * <li> This annotation can be used with the following annotations:
73 *      {@link XmlID},
74 *      {@link XmlIDREF},
75 *      {@link XmlList},
76 *      {@link XmlSchemaType},
77 *      {@link XmlValue},
78 *      {@link XmlAttachmentRef},
79 *      {@link XmlMimeType},
80 *      {@link XmlInlineBinaryData},
81 *      {@link javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter}.</li>
82 * </ul>
83 * </p>
84 *
85 * <p> <b>Example 1: </b>Map a JavaBean property to an XML attribute.</p>
86 * <pre>
87 * //Example: Code fragment
88 * public class USPrice {
89 *     &#64;XmlAttribute
90 *     public java.math.BigDecimal getPrice() {...} ;
91 *     public void setPrice(java.math.BigDecimal ) {...};
92 * }
93 *
94 * &lt;!-- Example: XML Schema fragment -->
95 * &lt;xs:complexType name="USPrice">
96 *     &lt;xs:sequence>
97 *     &lt;/xs:sequence>
98 *     &lt;xs:attribute name="price" type="xs:decimal"/>
99 * &lt;/xs:complexType>
100 * </pre>
101 *
102 * <p> <b>Example 2: </b>Map a JavaBean property to an XML attribute with anonymous type.</p>
103 * See Example 7 in {@link XmlType}.
104 *
105 * <p> <b>Example 3: </b>Map a JavaBean collection property to an XML attribute.</p>
106 * <pre>
107 * // Example: Code fragment
108 * class Foo {
109 *     ...
110 *     &#64;XmlAttribute List<Integer> items;
111 * }
112 *
113 * &lt;!-- Example: XML Schema fragment -->
114 * &lt;xs:complexType name="foo">
115 *     ...
116 *     &lt;xs:attribute name="items">
117 *         &lt;xs:simpleType>
118 *             &lt;xs:list itemType="xs:int"/>
119 *         &lt;/xs:simpleType>

```



```

120 *      </xs:complexType>
121 *
122 * </pre>
123 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
124 * @see XmlType
125 * @since JAXB2.0
126 */
127
128 @Retention(RUNTIME) @Target({FIELD, METHOD})
129 public @interface XmlAttribute {
130     /**
131      * Name of the XML Schema attribute. By default, the XML Schema
132      * attribute name is derived from the JavaBean property name.
133      *
134      */
135     String name() default "##default";
136
137     /**
138      * Specifies if the XML Schema attribute is optional or
139      * required. If true, then the JavaBean property is mapped to a
140      * XML Schema attribute that is required. Otherwise it is mapped
141      * to a XML Schema attribute that is optional.
142      *
143      */
144     boolean required() default false;
145
146     /**
147      * Specifies the XML target namespace of the XML Schema
148      * attribute.
149      *
150      */
151     String namespace() default "##default" ;
152 }

```

### 3.11 XmlElement.java

Secuencia 45: javax/xml/bind/annotation/XmlElement.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *

```

```

17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
29 import java.lang.annotation.Retention;
30 import java.lang.annotation.Target;
31
32 import static java.lang.annotation.ElementType.*;
33 import static java.lang.annotation.ElementType.PARAMETER;
34 import static java.lang.annotation.RetentionPolicy.*;
35
36 /**
37  * Maps a JavaBean property to a XML element derived from property name.
38  *
39  * <p> <b>Usage</b> </p>
40  * <p>
41  * <tt>@XmlElement</tt> annotation can be used with the following program
42  * elements:
43  * <ul>
44  * <li> a JavaBean property </li>
45  * <li> non static, non transient field </li>
46  * <li> within {@link XmlElements}
47  * <p>
48  *
49  * </ul>
50  *
51  * The usage is subject to the following constraints:
52  * <ul>
53  * <li> This annotation can be used with following annotations:
54  *      {@link XmlID},
55  *      {@link XmlIDREF},
56  *      {@link XmlList},
57  *      {@link XmlSchemaType},
58  *      {@link XmlValue},
59  *      {@link XmlAttachmentRef},
60  *      {@link XmlMimeType},
61  *      {@link XmlInlineBinaryData},
62  *      {@link XmlElementWrapper},
63  *      {@link XmlJavaTypeAdapter}</li>
64  * <li> if the type of JavaBean property is a collection type of
65  *      array, an indexed property, or a parameterized list, and
66  *      this annotation is used with {@link XmlElements} then,
67  *      <tt>@XmlElement.type()</tt> must be DEFAULT.class since the
68  *      collection item type is already known. </li>
69  * </ul>

```

```

70 *
71 * <p>
72 * A JavaBean property, when annotated with @XmlElement annotation
73 * is mapped to a local element in the XML Schema complex type to
74 * which the containing class is mapped.
75 *
76 * <p>
77 * <b>Example 1: </b> Map a public non static non final field to local
78 * element
79 * <pre>
80 *     //Example: Code fragment
81 *     public class USPrice {
82 *         &#64;XmlElement(name="itemprice")
83 *         public java.math.BigDecimal price;
84 *     }
85 *
86 *     &lt;!-- Example: Local XML Schema element -->
87 *     &lt;xs:complexType name="USPrice"/>
88 *     &lt;xs:sequence>
89 *         &lt;xs:element name="itemprice" type="xs:decimal" minOccurs="0"/>
90 *     &lt;/sequence>
91 *     &lt;/xs:complexType>
92 * </pre>
93 * <p>
94 *
95 * <b> Example 2: </b> Map a field to a nillable element.
96 * <pre>
97 *
98 *     //Example: Code fragment
99 *     public class USPrice {
100 *         &#64;XmlElement(nillable=true)
101 *         public java.math.BigDecimal price;
102 *     }
103 *
104 *     &lt;!-- Example: Local XML Schema element -->
105 *     &lt;xs:complexType name="USPrice">
106 *         &lt;xs:sequence>
107 *             &lt;xs:element name="price" type="xs:decimal" nillable="true" minOccurs="0"/>
108 *         &lt;/sequence>
109 *     &lt;/xs:complexType>
110 * </pre>
111 * <p>
112 * <b> Example 3: </b> Map a field to a nillable, required element.
113 * <pre>
114 *
115 *     //Example: Code fragment
116 *     public class USPrice {
117 *         &#64;XmlElement(nillable=true, required=true)
118 *         public java.math.BigDecimal price;
119 *     }
120 *
121 *     &lt;!-- Example: Local XML Schema element -->
122 *     &lt;xs:complexType name="USPrice">

```

```

123 *      <xs:sequence>
124 *      <xs:element name="price" type="xs:decimal" nillable="true" minOccurs="1"/>
125 *      </sequence>
126 *      </xs:complexType>
127 *    </pre>
128 * <p>
129 *
130 * <p> <b>Example 4: </b>Map a JavaBean property to an XML element
131 * with anonymous type.</p>
132 * <p>
133 * See Example 6 in @{@link XmlType}.
134 *
135 * <p>
136 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
137 * @since JAXB2.0
138 */
139
140 @Retention(RUNTIME) @Target({FIELD, METHOD, PARAMETER})
141 public @interface XmlElement {
142     /**
143      * Name of the XML Schema element.
144      * <p> If the value is "##default", then element name is derived from the
145      * JavaBean property name.
146      */
147     String name() default "##default";
148
149     /**
150      * Customize the element declaration to be nillable.
151      * <p>If nillable() is true, then the JavaBean property is
152      * mapped to a XML Schema nillable element declaration.
153      */
154     boolean nillable() default false;
155
156     /**
157      * Customize the element declaration to be required.
158      * <p>If required() is true, then Javabeen property is mapped to
159      * an XML schema element declaration with minOccurs="1".
160      * maxOccurs is "1" for a single valued property and "unbounded"
161      * for a multivalued property.
162      * <p>If required() is false, then the Javabeen property is mapped
163      * to XML Schema element declaration with minOccurs="0".
164      * maxOccurs is "1" for a single valued property and "unbounded"
165      * for a multivalued property.
166      */
167
168     boolean required() default false;
169
170     /**
171      * XML target namespace of the XML Schema element.
172      * <p>
173      * If the value is "##default", then the namespace is determined
174      * as follows:
175      * <ol>

```

```

176 * <li>
177 * If the enclosing package has {@link XmlSchema} annotation,
178 * and its {@link XmlSchema#elementFormDefault() elementFormDefault}
179 * is {@link XmlNsForm#QUALIFIED QUALIFIED}, then the namespace of
180 * the enclosing class.
181 *
182 * <li>
183 * Otherwise &#39;&#39; (which produces unqualified element in the default
184 * namespace.
185 * </ol>
186 */
187 String namespace() default "##default";
188
189 /**
190 * Default value of this element.
191 *
192 * <p>
193 * The <pre>'&u0000'</pre> value specified as a default of this annotation element
194 * is used as a poor-man's substitute for null to allow implementations
195 * to recognize the 'no default value' state.
196 */
197 String defaultValue() default "&u0000";
198
199 /**
200 * The Java class being referenced.
201 */
202 Class type() default DEFAULT.class;
203
204 /**
205 * Used in {@link XmlElement#type()} to
206 * signal that the type be inferred from the signature
207 * of the property.
208 */
209 static final class DEFAULT {}
210 }

```

### 3.12 XmlElementDecl.java

Secuencia 46: javax/xml/bind/annotation/XmlElementDecl.java

```

1 /*
2 * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4 *
5 *
6 *
7 *
8 *
9 *
10 *
11 *
12 *
13 *
14 *

```

```

15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.bind.annotation;
27
28  import java.lang.annotation.Retention;
29  import java.lang.annotation.Target;
30
31  import static java.lang.annotation.RetentionPolicy.RUNTIME;
32  import static java.lang.annotation.ElementType.METHOD;
33
34  /**
35   * Maps a factory method to a XML element.
36   *
37   * <p> <b>Usage</b> </p>
38   *
39   * The annotation creates a mapping between an XML schema element
40   * declaration and a <i> element factory method </i> that returns a
41   * JAXBElement instance representing the element
42   * declaration. Typically, the element factory method is generated
43   * (and annotated) from a schema into the ObjectFactory class in a
44   * Java package that represents the binding of the element
45   * declaration's target namespace. Thus, while the annotation syntax
46   * allows &#64;XmlElementDecl to be used on any method, semantically
47   * its use is restricted to annotation of element factory method.
48   *
49   * The usage is subject to the following constraints:
50   *
51   * <ul>
52   *   <li> The class containing the element factory method annotated
53   *     with &#64;XmlElementDecl must be marked with {@link
54   *     XmlRegistry}. </li>
55   *   <li> The element factory method must take one parameter
56   *     assignable to {@link Object}.</li>
57   * </ul>
58   *
59   * <p><b>Example 1: </b>Annotation on a factory method
60   * <pre>
61   *     // Example: code fragment
62   *     &#64;XmlRegistry
63   *     class ObjectFactory {
64   *         &#64;XmlElementDecl(name="foo")
65   *         JAXBElement<String> createFoo(String s) { ... }
66   *     }
67   * </pre>

```

```

68 * <pre>
69 *     <!-- XML input -->
70 *     <!--foo>string</foo>
71 *
72 *     // Example: code fragment corresponding to XML input
73 *     JAXBElement<String> o =
74 *     (JAXBElement<String>)unmarshaller.unmarshal(aboveDocument);
75 *     // print JAXBElement instance to show values
76 *     System.out.println(o.getName()); // prints "{}foo"
77 *     System.out.println(o.getValue()); // prints "string"
78 *     System.out.println(o.getValue().getClass()); // prints "java.lang.String"
79 *
80 *     <!-- Example: XML schema definition -->
81 *     <!--xs:element name="foo" type="xs:string"/>
82 * </pre>
83 *
84 * <p><b>Example 2: </b> Element declaration with non local scope
85 * <p>
86 * The following example illustrates the use of scope annotation
87 * parameter in binding of element declaration in schema derived
88 * code.
89 * <p>
90 * The following example may be replaced in a future revision of
91 * this javadoc.
92 *
93 * <pre>
94 *     <!-- Example: XML schema definition -->
95 *     <!--xs:schema>
96 *     <!--xs:complexType name="pea">
97 *     <!--xs:choice maxOccurs="unbounded">
98 *     <!--xs:element name="foo" type="xs:string"/>
99 *     <!--xs:element name="bar" type="xs:string"/>
100 *     <!--/xs:choice>
101 *     <!--/xs:complexType>
102 *     <!--xs:element name="foo" type="xs:int"/>
103 *     <!--/xs:schema>
104 * </pre>
105 * <pre>
106 *     // Example: expected default binding
107 *     class Pea {
108 *         &#64;XmlElementRefs({
109 *             &#64;XmlElementRef(name="foo",type=JAXBElement.class)
110 *             &#64;XmlElementRef(name="bar",type=JAXBElement.class)
111 *         })
112 *         List<JAXBElement<String>> fooOrBar;
113 *     }
114 *
115 *     &#64;XmlRegistry
116 *     class ObjectFactory {
117 *         &#64;XmlElementDecl(scope=Pea.class,name="foo")
118 *         JAXBElement<String> createPeaFoo(String s);
119 *
120 *         &#64;XmlElementDecl(scope=Pea.class,name="bar")

```

```

121 *      JAXBElement<String> createPeaBar(String s);
122 *
123 *      &#64;XmlElementDecl(name="foo")
124 *      JAXBElement<Integer> createFoo(Integer i);
125 *  }
126 *
127 * </pre>
128 * Without scope createFoo and createPeaFoo would become ambiguous
129 * since both of them map to a XML schema element with the same local
130 * name "foo".
131 *
132 * @see XmlRegistry
133 * @since JAXB 2.0
134 */
135 @Retention(RUNTIME)
136 @Target({METHOD})
137 public @interface XmlElementDecl {
138     /**
139      * scope of the mapping.
140      *
141      * <p>
142      * If this is not {@link XmlElementDecl.GLOBAL}, then this element
143      * declaration mapping is only active within the specified class.
144      */
145     Class scope() default GLOBAL.class;
146
147     /**
148      * namespace name of the XML element.
149      * <p>
150      * If the value is "##default", then the value is the namespace
151      * name for the package of the class containing this factory method.
152      *
153      * @see #name()
154      */
155     String namespace() default "##default";
156
157     /**
158      * local name of the XML element.
159      *
160      * <p>
161      * <b> Note to reviewers: </b> There is no default name; since
162      * the annotation is on a factory method, it is not clear that the
163      * method name can be derived from the factory method name.
164      * @see #namespace()
165      */
166     String name();
167
168     /**
169      * namespace name of a substitution group's head XML element.
170      * <p>
171      * This specifies the namespace name of the XML element whose local
172      * name is specified by <tt>substitutionHeadName()</tt>.
173      * <p>

```



```

174 * If <tt>substitutionHeadName()</tt> is "", then this
175 * value can only be "##default". But the value is ignored since
176 * since this element is not part of substitution group when the
177 * value of <tt>substitutionHeadName()</tt> is "".
178 * <p>
179 * If <tt>substitutionHeadName()</tt> is not "" and the value is
180 * "##default", then the namespace name is the namespace name to
181 * which the package of the containing class, marked with {@link
182 * XmlRegistry }, is mapped.
183 * <p>
184 * If <tt>substitutionHeadName()</tt> is not "" and the value is
185 * not "##default", then the value is the namespace name.
186 *
187 * @see #substitutionHeadName()
188 */
189 String substitutionHeadNamespace() default "##default";
190
191 /**
192 * XML local name of a substitution group's head element.
193 * <p>
194 * If the value is "", then this element is not part of any
195 * substitution group.
196 *
197 * @see #substitutionHeadNamespace()
198 */
199 String substitutionHeadName() default "";
200
201 /**
202 * Default value of this element.
203 *
204 * <p>
205 * The <pre>'u0000'</pre> value specified as a default of this annotation element
206 * is used as a poor-man's substitute for null to allow implementations
207 * to recognize the 'no default value' state.
208 */
209 String defaultValue() default "\u0000";
210
211 /**
212 * Used in {@link XmlElementDecl#scope()} to
213 * signal that the declaration is in the global scope.
214 */
215 public final class GLOBAL {}
216 }

```

### 3.13 XmlElementRef.java

Secuencia 47: javax/xml/bind/annotation/XmlElementRef.java

```

1 /*
2 * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4 *
5 *
6 *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
29 import java.lang.annotation.Retention;
30 import java.lang.annotation.Target;
31
32 import static java.lang.annotation.RetentionPolicy.RUNTIME;
33 import static java.lang.annotation.ElementType.FIELD;
34 import static java.lang.annotation.ElementType.METHOD;
35
36 /**
37  * <p>
38  * Maps a JavaBean property to a XML element derived from property's type.
39  * <p>
40  * <b>Usage</b>
41  * <p>
42  * <tt>XmlElementRef</tt> annotation can be used with a
43  * JavaBean property or from within {@link XmlElementRefs}
44  * <p>
45  * This annotation dynamically associates an XML element name with the JavaBean
46  * property. When a JavaBean property is annotated with {@link
47  * XmlElement}, the XML element name is statically derived from the
48  * JavaBean property name. However, when this annotation is used, the
49  * XML element name is derived from the instance of the type of the
50  * JavaBean property at runtime.
51  *
52  * <h3> XML Schema substitution group support </h3>
53  * XML Schema allows a XML document author to use XML element names
54  * that were not statically specified in the content model of a
55  * schema using substitution groups. Schema derived code provides
56  * support for substitution groups using an <i>element property</i>,
57  * (section 5.5.5, "Element Property" of JAXB 2.0 specification). An
58  * element property method signature is of the form:
59  * <pre>
```

```

60 *     public void setTerm(JAXBElement<? extends Operator>);
61 *     public JAXBElement<? extends Operator> getTerm();
62 * </pre>
63 * <p>
64 * An element factory method annotated with {@link XmlElementDecl} is
65 * used to create a <tt>JAXBElement</tt> instance, containing an XML
66 * element name. The presence of &#64;XmlElementRef annotation on an
67 * element property indicates that the element name from <tt>JAXBElement</tt>
68 * instance be used instead of deriving an XML element name from the
69 * JavaBean property name.
70 *
71 * <p>
72 * The usage is subject to the following constraints:
73 * <ul>
74 * <li> If the collection item type (for collection property) or
75 *     property type (for single valued property) is
76 *     {@link javax.xml.bind.JAXBElement}, then
77 *     <tt>&#64;XmlElementRef.name()</tt> and <tt>&#64;XmlElementRef.namespace()</tt> must
78 *     point an element factory method with an @XmlElementDecl
79 *     annotation in a class annotated with @XmlRegistry (usually
80 *     ObjectFactory class generated by the schema compiler) :
81 *     <ul>
82 *         <li> @XmlElementDecl.name() must equal @XmlElementRef.name() </li>
83 *         <li> @XmlElementDecl.namespace() must equal @XmlElementRef.namespace(). </li>
84 *     </ul>
85 * </li>
86 * <li> If the collection item type (for collection property) or
87 *     property type (for single valued property) is not
88 *     {@link javax.xml.bind.JAXBElement}, then the type referenced by the
89 *     property or field must be annotated with {@link XmlRootElement}. </li>
90 * <li> This annotation can be used with the following annotations:
91 *     {@link XmlElementWrapper}, {@link XmlJavaTypeAdapter}.
92 * </ul>
93 *
94 * <p>See "Package Specification" in javax.xml.bind.package javadoc for
95 * additional common information.</p>
96 *
97 * <p><b>Example 1: Ant Task Example</b></p>
98 * The following Java class hierarchy models an Ant build
99 * script. An Ant task corresponds to a class in the class
100 * hierarchy. The XML element name of an Ant task is indicated by the
101 * &#64;XmlRootElement annotation on its corresponding class.
102 * <pre>
103 *     &#64;XmlRootElement(name="target")
104 *     class Target {
105 *         // The presence of &#64;XmlElementRef indicates that the XML
106 *         // element name will be derived from the &#64;XmlRootElement
107 *         // annotation on the type (for e.g. "jar" for JarTask).
108 *         &#64;XmlElementRef
109 *         List<Task> tasks;
110 *     }
111 *
112 *     abstract class Task {

```

```

113 *     }
114 *
115 *     &#64;XmlRootElement(name="jar")
116 *     class JarTask extends Task {
117 *         ...
118 *     }
119 *
120 *     &#64;XmlRootElement(name="javac")
121 *     class JavacTask extends Task {
122 *         ...
123 *     }
124 *
125 *     &lt;!-- XML Schema fragment -->
126 *     &lt;xs:element name="target" type="Target">
127 *     &lt;xs:complexType name="Target">
128 *     &lt;xs:sequence>
129 *     &lt;xs:choice maxOccurs="unbounded">
130 *     &lt;xs:element ref="jar">
131 *     &lt;xs:element ref="javac">
132 *     &lt;/xs:choice>
133 *     &lt;/xs:sequence>
134 *     &lt;/xs:complexType>
135 *
136 * </pre>
137 * <p>
138 * Thus the following code fragment:
139 * <pre>
140 *     Target target = new Target();
141 *     target.tasks.add(new JarTask());
142 *     target.tasks.add(new JavacTask());
143 *     marshal(target);
144 * </pre>
145 * will produce the following XML output:
146 * <pre>
147 *     &lt;target>
148 *     &lt;jar>
149 *     ....
150 *     &lt;/jar>
151 *     &lt;javac>
152 *     ....
153 *     &lt;/javac>
154 *     &lt;/target>
155 * </pre>
156 * <p>
157 * It is not an error to have a class that extends <tt>Task</tt>
158 * that doesn't have {<link XmlRootElement>}. But they can't show up in an
159 * XML instance (because they don't have XML element names).
160 *
161 * <p><b>Example 2: XML Schema Substitution group support</b>
162 * <p> The following example shows the annotations for XML Schema
163 * substitution groups. The annotations and the ObjectFactory are
164 * derived from the schema.
165 *

```

```

166 * <pre>
167 *      &#64;XmlElement
168 *      class Math {
169 *          // The value of {@link #type()}is
170 *          // JAXBElement.class , which indicates the XML
171 *          // element name ObjectFactory - in general a class marked
172 *          // with &#64;XmlRegistry. (See ObjectFactory below)
173 *          //
174 *          // The {@link #name()} is "operator", a pointer to a
175 *          // factory method annotated with a
176 *          // {@link XmlElementDecl} with the name "operator". Since
177 *          // "operator" is the head of a substitution group that
178 *          // contains elements "add" and "sub" elements, "operator"
179 *          // element can be substituted in an instance document by
180 *          // elements "add" or "sub". At runtime, JAXBElement
181 *          // instance contains the element name that has been
182 *          // substituted in the XML document.
183 *          //
184 *          &#64;XmlElementRef(type=JAXBElement.class,name="operator")
185 *          JAXBElement<? extends Operator> term;
186 *      }
187 *
188 *      &#64;XmlRegistry
189 *      class ObjectFactory {
190 *          &#64;XmlElementDecl(name="operator")
191 *          JAXBElement<Operator> createOperator(Operator o) {...}
192 *          &#64;XmlElementDecl(name="add",substitutionHeadName="operator")
193 *          JAXBElement<Operator> createAdd(Operator o) {...}
194 *          &#64;XmlElementDecl(name="sub",substitutionHeadName="operator")
195 *          JAXBElement<Operator> createSub(Operator o) {...}
196 *      }
197 *
198 *      class Operator {
199 *          ...
200 *      }
201 * </pre>
202 * <p>
203 * Thus, the following code fragment
204 * <pre>
205 *      Math m = new Math();
206 *      m.term = new ObjectFactory().createAdd(new Operator());
207 *      marshal(m);
208 * </pre>
209 * will produce the following XML output:
210 * <pre>
211 *      <math>
212 *          <add>...</add>
213 *      </math>
214 * </pre>
215 *
216 *
217 * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems,Inc. </li><li>Sekhar Vajjhala, Sun
      Microsystems, Inc.</li></ul>

```

```

218 * @see XmlElementRefs
219 * @since JAXB2.0
220 */
221 @Retention(RUNTIME)
222 @Target({FIELD,METHOD})
223 public @interface XmlElementRef {
224     /**
225      * The Java type being referenced.
226      * <p>
227      * If the value is DEFAULT.class, the type is inferred from the
228      * the type of the JavaBean property.
229      */
230     Class type() default DEFAULT.class;
231
232     /**
233      * This parameter and {@link #name()} are used to determine the
234      * XML element for the JavaBean property.
235      *
236      * <p> If <tt>type()</tt> is <tt>JAXBElement.class</tt> , then
237      * <tt>namespace()</tt> and <tt>name()</tt>
238      * point to a factory method with {@link XmlElementDecl}. The XML
239      * element name is the element name from the factory method's
240      * {@link XmlElementDecl} annotation or if an element from its
241      * substitution group (of which it is a head element) has been
242      * substituted in the XML document, then the element name is from the
243      * {@link XmlElementDecl} on the substituted element.
244      *
245      * <p> If {@link #type()} is not <tt>JAXBElement.class</tt>, then
246      * the XML element name is the XML element name statically
247      * associated with the type using the annotation {@link
248      * XmlRootElement} on the type. If the type is not annotated with
249      * an {@link XmlElementDecl}, then it is an error.
250      *
251      * <p> If <tt>type()</tt> is not <tt>JAXBElement.class</tt>, then
252      * this value must be "".
253      */
254     String namespace() default "";
255
256     /**
257      *
258      * @see #namespace()
259      */
260     String name() default "##default";
261
262     /**
263      * Used in {@link XmlElementRef#type()} to
264      * signal that the type be inferred from the signature
265      * of the property.
266      */
267     static final class DEFAULT {}
268
269     /**
270      * Customize the element declaration to be required.

```

```

271     * <p>
272     * If required() is true, then Javabean property is mapped to
273     * an XML schema element declaration with minOccurs="1".
274     * maxOccurs is "1" for a single valued property and "unbounded"
275     * for a multivalued property.
276     *
277     * <p>
278     * If required() is false, then the Javabean property is mapped
279     * to XML Schema element declaration with minOccurs="0".
280     * maxOccurs is "1" for a single valued property and "unbounded"
281     * for a multivalued property.
282     *
283     * <p>
284     * For compatibility with JAXB 2.1, this property defaults to <tt>true</tt>,
285     * despite the fact that {@link XmlElement#required()} defaults to false.
286     *
287     * @since 2.2
288     */
289     boolean required() default true;
290 }

```

### 3.14 XmlElementRefs.java

Secuencia 48: javax/xml/bind/annotation/XmlElementRefs.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
29 import java.lang.annotation.Retention;

```

```

30 import java.lang.annotation.Target;
31 import static java.lang.annotation.ElementType.FIELD;
32 import static java.lang.annotation.ElementType.METHOD;
33 import static java.lang.annotation.RetentionPolicy.RUNTIME;
34
35 /**
36  * Marks a property that refers to classes with {@link XmlElement}
37  * or JAXBElement.
38  *
39  * <p>
40  * Compared to an element property (property with {@link XmlElement}
41  * annotation), a reference property has a different substitution semantics.
42  * When a sub-class is assigned to a property, an element property produces
43  * the same tag name with @xsi:type, whereas a reference property produces
44  * a different tag name (the tag name that's on the the sub-class.)
45  *
46  * <p> This annotation can be used with the following annotations:
47  * {@link XmlJavaTypeAdapter}, {@link XmlElementWrapper}.
48  *
49  * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Sekhar Vajjhala, Sun
50  *     Microsystems, Inc.</li></ul>
51  *
52  * @see XmlElementWrapper
53  * @see XmlElementRef
54  * @since JAXB2.0
55  */
56 @Retention(RUNTIME)
57 @Target({FIELD,METHOD})
58 public @interface XmlElementRefs {
59     XmlElementRef[] value();
60 }

```

### 3.15 XmlElementWrapper.java

Secuencia 49: javax/xml/bind/annotation/XmlElementWrapper.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *

```



```

19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
29 import static java.lang.annotation.RetentionPolicy.RUNTIME;
30 import static java.lang.annotation.ElementType.FIELD;
31 import static java.lang.annotation.ElementType.METHOD;
32 import java.lang.annotation.Retention;
33 import java.lang.annotation.Target;
34
35 /**
36  * Generates a wrapper element around XML representation.
37  *
38  * This is primarily intended to be used to produce a wrapper
39  * XML element around collections. The annotation therefore supports
40  * two forms of serialization shown below.
41  *
42  * <pre>
43  *     //Example: code fragment
44  *     int[] names;
45  *
46  *     // XML Serialization Form 1 (Unwrapped collection)
47  *     <names> ... </names>
48  *     <names> ... </names>
49  *
50  *     // XML Serialization Form 2 ( Wrapped collection )
51  *     <wrapperElement>
52  *         <names> value-of-item </names>
53  *         <names> value-of-item </names>
54  *         ....
55  *     </wrapperElement>
56  * </pre>
57  *
58  * <p> The two serialized XML forms allow a null collection to be
59  * represented either by absence or presence of an element with a
60  * nillable attribute.
61  *
62  * <p> <b>Usage</b> </p>
63  * <p>
64  * The <tt>@XmlElementWrapper</tt> annotation can be used with the
65  * following program elements:
66  * <ul>
67  * <li> JavaBean property </li>
68  * <li> non static, non transient field </li>
69  * </ul>
70  *
71  * <p>The usage is subject to the following constraints:

```

```

72 * <ul>
73 *   <li> The property must be a collection property </li>
74 *   <li> This annotation can be used with the following annotations:
75 *       {@link XmlElement},
76 *       {@link XmlElements},
77 *       {@link XmlElementRef},
78 *       {@link XmlElementRefs},
79 *       {@link XmlJavaTypeAdapter}.</li>
80 * </ul>
81 *
82 * <p>See "Package Specification" in javax.xml.bind.package javadoc for
83 * additional common information.</p>
84 *
85 * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Sekhar Vajjhala, Sun
86 *   Microsystems, Inc.</li></ul>
87 * @see XmlElement
88 * @see XmlElements
89 * @see XmlElementRef
90 * @see XmlElementRefs
91 * @since JAXB2.0
92 */
93
94 @Retention(RUNTIME) @Target({FIELD, METHOD})
95 public @interface XmlElementWrapper {
96     /**
97      * Name of the XML wrapper element. By default, the XML wrapper
98      * element name is derived from the JavaBean property name.
99      */
100     String name() default "##default";
101
102     /**
103      * XML target namespace of the XML wrapper element.
104      * <p>
105      * If the value is "##default", then the namespace is determined
106      * as follows:
107      * <ol>
108      * <li>
109      * If the enclosing package has {@link XmlSchema} annotation,
110      * and its {@link XmlSchema#elementFormDefault\(\) elementFormDefault}
111      * is {@link XmlNsForm#QUALIFIED QUALIFIED}, then the namespace of
112      * the enclosing class.
113      *
114      * <li>
115      * Otherwise "" (which produces unqualified element in the default
116      * namespace.
117      * </ol>
118      */
119     String namespace() default "##default";
120
121     /**
122      * If true, the absence of the collection is represented by
123      * using <tt>xsi:nil='true'</tt>. Otherwise, it is represented by

```

```

124     * the absence of the element.
125     */
126     boolean nillable() default false;
127
128     /**
129     * Customize the wrapper element declaration to be required.
130     *
131     * <p>
132     * If required() is true, then the corresponding generated
133     * XML schema element declaration will have <tt>minOccurs="1"</tt>,
134     * to indicate that the wrapper element is always expected.
135     *
136     * <p>
137     * Note that this only affects the schema generation, and
138     * not the unmarshalling or marshalling capability. This is
139     * simply a mechanism to let users express their application constraints
140     * better.
141     *
142     * @since JAXB 2.1
143     */
144     boolean required() default false;
145 }

```

### 3.16 XmlElements.java

Secuencia 50: javax/xml/bind/annotation/XmlElements.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27

```

```

28 import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
29 import static java.lang.annotation.RetentionPolicy.RUNTIME;
30 import static java.lang.annotation.ElementType.FIELD;
31 import static java.lang.annotation.ElementType.METHOD;
32 import java.lang.annotation.Retention;
33 import java.lang.annotation.Target;
34
35 /**
36  * <p>
37  * A container for multiple {@link XmlElement} annotations.
38  *
39  * Multiple annotations of the same type are not allowed on a program
40  * element. This annotation therefore serves as a container annotation
41  * for multiple XmlElements as follows:
42  *
43  * <pre>
44  * XmlElement({ @XmlElement(...),@XmlElement(...) })
45  * </pre>
46  *
47  * <p>The @XmlElement annotation can be used with the
48  * following program elements: </p>
49  * <ul>
50  * <li> a JavaBean property </li>
51  * <li> non static, non transient field </li>
52  * </ul>
53  *
54  * This annotation is intended for annotation a JavaBean collection
55  * property (e.g. List).
56  *
57  * <p><b>Usage</b></p>
58  *
59  * <p>The usage is subject to the following constraints:
60  * <ul>
61  * <li> This annotation can be used with the following
62  * annotations: @link XmlIDREF, @link XmlElementWrapper. </li>
63  * <li> If @XmlIDREF is also specified on the JavaBean property,
64  * then each XmlElement.type() must contain a JavaBean
65  * property annotated with <tt>XmlID</tt>.</li>
66  * </ul>
67  *
68  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
69  * additional common information.</p>
70  *
71  * <hr>
72  *
73  * <p><b>Example 1:</b> Map to a list of elements</p>
74  * <pre>
75  *
76  * // Mapped code fragment
77  * public class Foo {
78  *     XmlElements(
79  *         XmlElement(name="A", type=Integer.class),
80  *         XmlElement(name="B", type=Float.class)

```

```

81 *     }
82 *     public List items;
83 * }
84 *
85 * <!-- XML Representation for a List of {1,2.5}
86 *     XML output is not wrapped using another element -->
87 * ...
88 * <A> 1 </A>
89 * <B> 2.5 </B>
90 * ...
91 *
92 * <!-- XML Schema fragment -->
93 * <xs:complexType name="Foo">
94 *     <xs:sequence>
95 *         <xs:choice minOccurs="0" maxOccurs="unbounded">
96 *             <xs:element name="A" type="xs:int"/>
97 *             <xs:element name="B" type="xs:float"/>
98 *         </xs:choice>
99 *     </xs:sequence>
100 * </xs:complexType>
101 *
102 * </pre>
103 *
104 * <p><b>Example 2:</b> Map to a list of elements wrapped with another element
105 * </p>
106 * <pre>
107 *
108 * // Mapped code fragment
109 * public class Foo {
110 *     &#64;XmlElementWrapper(name="bar")
111 *     &#64;XmlElements(
112 *         &#64;XmlElement(name="A", type=Integer.class),
113 *         &#64;XmlElement(name="B", type=Float.class)
114 *     )
115 *     public List items;
116 * }
117 *
118 * <!-- XML Schema fragment -->
119 * <xs:complexType name="Foo">
120 *     <xs:sequence>
121 *         <xs:element name="bar">
122 *             <xs:complexType>
123 *                 <xs:choice minOccurs="0" maxOccurs="unbounded">
124 *                     <xs:element name="A" type="xs:int"/>
125 *                     <xs:element name="B" type="xs:float"/>
126 *                 </xs:choice>
127 *             </xs:complexType>
128 *         </xs:element>
129 *     </xs:sequence>
130 * </xs:complexType>
131 * </pre>
132 *
133 * <p><b>Example 3:</b> Change element name based on type using an adapter.

```

```

134 * </p>
135 * <pre>
136 *     class Foo {
137 *         &#64;XmlJavaTypeAdapter(QtoPAdapter.class)
138 *         &#64;XmlElements({
139 *             &#64;XmlElement(name="A",type=PX.class),
140 *             &#64;XmlElement(name="B",type=PY.class)
141 *         })
142 *         Q bar;
143 *     }
144 *
145 *     &#64;XmlType abstract class P {...}
146 *     &#64;XmlType(name="PX") class PX extends P {...}
147 *     &#64;XmlType(name="PY") class PY extends P {...}
148 *
149 *     &lt;!-- XML Schema fragment -->
150 *     &lt;xs:complexType name="Foo">
151 *         &lt;xs:sequence>
152 *             &lt;xs:element name="bar">
153 *                 &lt;xs:complexType>
154 *                     &lt;xs:choice minOccurs="0" maxOccurs="unbounded">
155 *                         &lt;xs:element name="A" type="PX"/>
156 *                         &lt;xs:element name="B" type="PY"/>
157 *                     &lt;/xs:choice>
158 *                 &lt;/xs:complexType>
159 *             &lt;/xs:element>
160 *         &lt;/xs:sequence>
161 *     &lt;/xs:complexType>
162 * </pre>
163 *
164 * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Sekhar Vajjhala, Sun
165 *     Microsystems, Inc.</li></ul>
166 * @see XmlElement
167 * @see XmlElementRef
168 * @see XmlElementRefs
169 * @see XmlJavaTypeAdapter
170 * @since JAXB2.0
171 */
172 @Retention(RUNTIME) @Target({FIELD,METHOD})
173 public @interface XmlElements {
174     /**
175      * Collection of {@link XmlElement} annotations
176      */
177     XmlElement[] value();
178 }

```

### 3.17 XmlEnum.java

Secuencia 51: javax/xml/bind/annotation/XmlEnum.java

```

1 /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *

```

```
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import static java.lang.annotation.ElementType.TYPE;
29 import java.lang.annotation.Retention;
30 import static java.lang.annotation.RetentionPolicy.RUNTIME;
31 import java.lang.annotation.Target;
32
33 /**
34  * <p>
35  * Maps an enum type {@link Enum} to XML representation.
36  *
37  * <p>This annotation, together with {@link XmlEnumValue} provides a
38  * mapping of enum type to XML representation.
39  *
40  * <p> <b>Usage</b> </p>
41  * <p>
42  * The <tt>@XmlEnum</tt> annotation can be used with the
43  * following program elements:
44  * <ul>
45  * <li>enum type</li>
46  * </ul>
47  *
48  * <p> The usage is subject to the following constraints:
49  * <ul>
50  * <li> This annotation can be used the following other annotations:
51  * <ul>
52  * <li>{@link XmlType},
53  * <li>{@link XmlRootElement} </li>
54  * </ul>
55  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
56  * additional common information </p>
57  *
58  * <p>An enum type is mapped to a schema simple type with enumeration
```

```

58  * facets. The schema type is derived from the Java type to which
59  * <tt>@XmlEnum.value()</tt>. Each enum constant <tt>@XmlEnumValue</tt>
60  * must have a valid lexical representation for the type
61  * <tt>@XmlEnum.value()</tt> .
62  *
63  * <p><b>Examples:</b> See examples in {@link XmlEnumValue}
64  *
65  * @since JAXB2.0
66  */
67
68  @Retention(RUNTIME) @Target({TYPE})
69  public @interface XmlEnum {
70      /**
71       * Java type that is mapped to a XML simple type.
72       *
73       */
74      Class<?> value() default String.class;
75  }

```

### 3.18 XmlEnumValue.java

Secuencia 52: javax/xml/bind/annotation/XmlEnumValue.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30 import static java.lang.annotation.RetentionPolicy.RUNTIME;
31 import static java.lang.annotation.ElementType.FIELD;

```



```

32
33 /**
34  * Maps an enum constant in {@link Enum} type to XML representation.
35  *
36  * <p> <b>Usage</b> </p>
37  *
38  * <p> The <tt>@XmlEnumValue</tt> annotation can be used with the
39  *     following program elements:
40  * <ul>
41  *   <li>enum constant</li>
42  * </ul>
43  *
44  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
45  * additional common information.</p>
46  *
47  * <p>This annotation, together with {@link XmlEnum} provides a
48  * mapping of enum type to XML representation.
49  *
50  * <p>An enum type is mapped to a schema simple type with enumeration
51  * facets. The schema type is derived from the Java type specified in
52  * <tt>@XmlEnum.value()</tt>. Each enum constant <tt>@XmlEnumValue</tt>
53  * must have a valid lexical representation for the type
54  * <tt>@XmlEnum.value()</tt>
55  *
56  * <p> In the absence of this annotation, {@link Enum#name()} is used
57  * as the XML representation.
58  *
59  * <p> <b>Example 1: </b>Map enum constant name -> enumeration facet</p>
60  * <pre>
61  *     //Example: Code fragment
62  *     &#64;XmlEnum(String.class)
63  *     public enum Card { CLUBS, DIAMONDS, HEARTS, SPADES }
64  *
65  *     &lt;!-- Example: XML Schema fragment -->
66  *     &lt;xs:simpleType name="Card">
67  *       &lt;xs:restriction base="xs:string"/>
68  *       &lt;xs:enumeration value="CLUBS"/>
69  *       &lt;xs:enumeration value="DIAMONDS"/>
70  *       &lt;xs:enumeration value="HEARTS"/>
71  *       &lt;xs:enumeration value="SPADES"/>
72  *     &lt;/xs:simpleType>
73  * </pre>
74  *
75  * <p><b>Example 2: </b>Map enum constant name(value) -> enumeration facet </p>
76  * <pre>
77  *     //Example: code fragment
78  *     &#64;XmlType
79  *     &#64;XmlEnum(Integer.class)
80  *     public enum Coin {
81  *       &#64;XmlEnumValue("1") PENNY(1),
82  *       &#64;XmlEnumValue("5") NICKEL(5),
83  *       &#64;XmlEnumValue("10") DIME(10),
84  *       &#64;XmlEnumValue("25") QUARTER(25) }

```

```

85  *
86  *      <!-- Example: XML Schema fragment -->
87  *      <xs:simpleType name="Coin">
88  *          <xs:restriction base="xs:int">
89  *              <xs:enumeration value="1"/>
90  *              <xs:enumeration value="5"/>
91  *              <xs:enumeration value="10"/>
92  *              <xs:enumeration value="25"/>
93  *          </xs:restriction>
94  *      </xs:simpleType>
95  * </pre>
96  *
97  * <p><b>Example 3: </b>Map enum constant name -> enumeration facet </p>
98  *
99  * <pre>
100 *      //Code fragment
101 *      &#64;XmlType
102 *      &#64;XmlEnum(Integer.class)
103 *      public enum Code {
104 *          &#64;XmlEnumValue("1") ONE,
105 *          &#64;XmlEnumValue("2") TWO;
106 *      }
107 *
108 *      <!-- Example: XML Schema fragment -->
109 *      <xs:simpleType name="Code">
110 *          <xs:restriction base="xs:int">
111 *              <xs:enumeration value="1"/>
112 *              <xs:enumeration value="2"/>
113 *          </xs:restriction>
114 *      </xs:simpleType>
115 * </pre>
116 *
117 * @since JAXB 2.0
118 */
119 @Retention(RUNTIME)
120 @Target({FIELD})
121 public @interface XmlEnumValue {
122     String value();
123 }

```

### 3.19 XmlID.java

Secuencia 53: javax/xml/bind/annotation/XmlID.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```

```

11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Target;
29 import java.lang.annotation.Retention;
30 import static java.lang.annotation.ElementType.*;
31 import static java.lang.annotation.RetentionPolicy.*;
32
33 /**
34  * <p>
35  * Maps a JavaBean property to XML ID.
36  *
37  * <p>
38  * To preserve referential integrity of an object graph across XML
39  * serialization followed by a XML deserialization, requires an object
40  * reference to be marshalled by reference or containment
41  * appropriately. Annotations <tt>#64;XmlID</tt> and <tt>#64;XmlIDREF</tt>
42  * together allow a customized mapping of a JavaBean property's
43  * type by containment or reference.
44  *
45  * <p><b>Usage</b></p>
46  * The <tt>#64;XmlID</tt> annotation can be used with the following
47  * program elements:
48  * <ul>
49  * <li> a JavaBean property </li>
50  * <li> non static, non transient field </li>
51  * </ul>
52  *
53  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
54  * additional common information.</p>
55  *
56  * The usage is subject to the following constraints:
57  * <ul>
58  * <li> At most one field or property in a class can be annotated
59  * with <tt>#64;XmlID</tt>. </li>
60  * <li> The JavaBean property's type must be <tt>java.lang.String</tt>.</li>
61  * <li> The only other mapping annotations that can be used
62  * with <tt>#64;XmlID</tt>
63  * are: <tt>#64;XmlElement</tt> and <tt>#64;XmlAttribute</tt>.</li>

```

```

64 * </ul>
65 *
66 * <p><b>Example</b>: Map a JavaBean property's type to <tt>xs:ID</tt></p>
67 * <pre>
68 *     // Example: code fragment
69 *     public class Customer {
70 *         &#64;XmlAttribute
71 *         &#64;XmlID
72 *         public String getCustomerID();
73 *         public void setCustomerID(String id);
74 *         .... other properties not shown
75 *     }
76 *
77 *     &lt;!-- Example: XML Schema fragment -->
78 *     &lt;xs:complexType name="Customer">
79 *         &lt;xs:complexContent>
80 *             &lt;xs:sequence>
81 *                 ....
82 *             &lt;/xs:sequence>
83 *             &lt;xs:attribute name="customerID" type="xs:ID"/>
84 *         &lt;/xs:complexContent>
85 *     &lt;/xs:complexType>
86 * </pre>
87 *
88 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
89 * @see XmlIDREF
90 * @since JAXB2.0
91 */
92
93 @Retention(RUNTIME) @Target({FIELD, METHOD})
94 public @interface XmlID { }

```

### 3.20 XmlIDREF.java

Secuencia 54: javax/xml/bind/annotation/XmlIDREF.java

```

1 /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *

```

```

19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Target;
29 import java.lang.annotation.Retention;
30 import static java.lang.annotation.ElementType.*;
31 import static java.lang.annotation.RetentionPolicy.*;
32
33 /**
34  * <p>
35  * Maps a JavaBean property to XML IDREF.
36  *
37  * <p>
38  * To preserve referential integrity of an object graph across XML
39  * serialization followed by a XML deserialization, requires an object
40  * reference to be marshalled by reference or containment
41  * appropriately. Annotations <tt>#64;XmlID</tt> and <tt>#64;XmlIDREF</tt>
42  * together allow a customized mapping of a JavaBean property's
43  * type by containment or reference.
44  *
45  * <p><b>Usage</b></p>
46  * The <tt>#64;XmlIDREF</tt> annotation can be used with the following
47  * program elements:
48  * <ul>
49  * <li> a JavaBean property </li>
50  * <li> non static, non transient field </li>
51  * </ul>
52  *
53  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
54  * additional common information.</p>
55  *
56  * <p>The usage is subject to the following constraints:
57  * <ul>
58  *
59  * <li> If the type of the field or property is a collection type,
60  * then the collection item type must contain a property or
61  * field annotated with <tt>#64;XmlID</tt>. </li>
62  * <li> If the field or property is single valued, then the type of
63  * the property or field must contain a property or field
64  * annotated with <tt>#64;XmlID</tt>.
65  * <p>Note: If the collection item type or the type of the
66  * property (for non collection type) is java.lang.Object, then
67  * the instance must contain a property/field annotated with
68  * <tt>#64;XmlID</tt> attribute.
69  * </li>
70  * <li> This annotation can be used with the following annotations:
71  * {@link XmlElement}, {@link XmlAttribute}, {@link XmlList},

```

```

72 *      and {@link XmlElements}.

```

```

125 * public class Shipping {
126 *
127 *     // map by reference
128 *     &#64;XmlIDREF public Customer getCustomer();
129 *     public void setCustomer(Customer customer);
130 * }
131 *
132 * // at least one class must reference Customer by containment;
133 * // Customer instances won't be marshalled.
134 * &#64;XmlElement(name="CustomerData")
135 * public class CustomerData {
136 *     // map reference to Customer by containment by default.
137 *     public Customer getCustomer();
138 *
139 *     // maps reference to Shipping by containment by default.
140 *     public Shipping getShipping();
141 *
142 *     // maps reference to Invoice by containment by default.
143 *     public Invoice getInvoice();
144 * }
145 *
146 * &lt;!-- XML Schema mapping for above code frament -->
147 *
148 * &lt;xs:complexType name="Invoice">
149 *     &lt;xs:complexContent>
150 *         &lt;xs:sequence>
151 *             &lt;xs:element name="customer" type="xs:IDREF"/>
152 *             ....
153 *         &lt;/xs:sequence>
154 *     &lt;/xs:complexContent>
155 * &lt;/xs:complexType>
156 *
157 * &lt;xs:complexType name="Shipping">
158 *     &lt;xs:complexContent>
159 *         &lt;xs:sequence>
160 *             &lt;xs:element name="customer" type="xs:IDREF"/>
161 *             ....
162 *         &lt;/xs:sequence>
163 *     &lt;/xs:complexContent>
164 * &lt;/xs:complexType>
165 *
166 * &lt;xs:complexType name="Customer">
167 *     &lt;xs:complexContent>
168 *         &lt;xs:sequence>
169 *             ....
170 *         &lt;/xs:sequence>
171 *         &lt;xs:attribute name="CustomerID" type="xs:ID"/>
172 *     &lt;/xs:complexContent>
173 * &lt;/xs:complexType>
174 *
175 * &lt;xs:complexType name="CustomerData">
176 *     &lt;xs:complexContent>
177 *         &lt;xs:sequence>

```

```

178 *      <xs:element name="customer" type="xs:Customer"/>
179 *      <xs:element name="shipping" type="xs:Shipping"/>
180 *      <xs:element name="invoice" type="xs:Invoice"/>
181 *      </xs:sequence>
182 *      </xs:complexContent>
183 *      </xs:complexType>
184 *
185 *      <xs:element name="customerData" type="xs:CustomerData"/>
186 *
187 *      <!-- Instance document conforming to the above XML Schema -->
188 *      <customerData>
189 *          <customer customerID="Alice">
190 *              ....
191 *          </customer>
192 *
193 *          <shipping customer="Alice">
194 *              ....
195 *          </shipping>
196 *
197 *          <invoice customer="Alice">
198 *              ....
199 *          </invoice>
200 *      </customerData>
201 *
202 * </pre>
203 *
204 * <p><b>Example 3: </b> Mapping List to repeating element of type IDREF
205 * <pre>
206 *      // Code fragment
207 *      public class Shipping {
208 *          &#64;XmlIDREF
209 *          &#64;XmlElement(name="Alice")
210 *          public List customers;
211 *      }
212 *
213 *      <!-- XML schema fragment -->
214 *      <xs:complexType name="Shipping">
215 *          <xs:sequence>
216 *              <xs:choice minOccurs="0" maxOccurs="unbounded">
217 *                  <xs:element name="Alice" type="xs:IDREF"/>
218 *              </xs:choice>
219 *          </xs:sequence>
220 *      </xs:complexType>
221 * </pre>
222 *
223 * <p><b>Example 4: </b> Mapping a List to a list of elements of type IDREF.
224 * <pre>
225 *      //Code fragment
226 *      public class Shipping {
227 *          &#64;XmlIDREF
228 *          &#64;XmlElements(
229 *              &#64;XmlElement(name="Alice", type="Customer.class")
230 *              &#64;XmlElement(name="John", type="InternationalCustomer.class")

```



```

231 *      public List customers;
232 *  }
233 *
234 *      <!-- XML Schema fragment -->
235 *      <xs:complexType name="Shipping">
236 *          <xs:sequence>
237 *              <xs:choice minOccurs="0" maxOccurs="unbounded">
238 *                  <xs:element name="Alice" type="xs:IDREF"/>
239 *                  <xs:element name="John" type="xs:IDREF"/>
240 *              </xs:choice>
241 *          </xs:sequence>
242 *      </xs:complexType>
243 * </pre>
244 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
245 * @see XmlID
246 * @since JAXB2.0
247 */
248
249 @Retention(RUNTIME) @Target({FIELD, METHOD})
250 public @interface XmlIDREF {}

```

### 3.21 XmlInlineBinaryData.java

Secuencia 55: javax/xml/bind/annotation/XmlInlineBinaryData.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;

```

```

30 import static java.lang.annotation.RetentionPolicy.RUNTIME;
31 import static java.lang.annotation.ElementType.FIELD;
32 import static java.lang.annotation.ElementType.METHOD;
33 import static java.lang.annotation.ElementType.TYPE;
34
35 import javax.xml.transform.Source;
36 import javax.xml.bind.attachment.AttachmentMarshaller;
37 import javax.activation.DataHandler;
38
39 /**
40  * Disable consideration of XOP encoding for datatypes that are bound to
41  * base64-encoded binary data in XML.
42  *
43  * <p>
44  * When XOP encoding is enabled as described in {@link AttachmentMarshaller#isXOPPackage()}, this
45  * annotation disables datatypes such as {@link java.awt.Image} or {@link Source} or <tt>byte[]</
46  * tt> that are bound to base64-encoded binary from being considered for
47  * XOP encoding. If a JAXB property is annotated with this annotation or if
48  * the JAXB property's base type is annotated with this annotation,
49  * neither
50  * {@link AttachmentMarshaller#addMtomAttachment(DataHandler, String, String)}
51  * nor
52  * {@link AttachmentMarshaller#addMtomAttachment(byte[], int, int, String, String, String)} is
53  * ever called for the property. The binary data will always be inlined.
54  *
55  * @author Joseph Fialli
56  * @since JAXB2.0
57  */
58 @Retention(RUNTIME)
59 @Target({FIELD, METHOD, TYPE})
60 public @interface XmlInlineBinaryData {
61 }

```

### 3.22 XmlList.java

Secuencia 56: javax/xml/bind/annotation/XmlList.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30 import static java.lang.annotation.RetentionPolicy.RUNTIME;
31 import static java.lang.annotation.ElementType.FIELD;
32 import static java.lang.annotation.ElementType.METHOD;
33 import static java.lang.annotation.ElementType.PARAMETER;
34
35 /**
36  * Used to map a property to a list simple type.
37  *
38  * <p><b>Usage</b></p>
39  * <p>
40  * The <tt>@XmlList</tt> annotation can be used with the
41  * following program elements:
42  * <ul>
43  *   <li> JavaBean property </li>
44  *   <li> field </li>
45  * </ul>
46  *
47  * <p>
48  * When a collection property is annotated just with @XmlElement,
49  * each item in the collection will be wrapped by an element.
50  * For example,
51  *
52  * <pre>
53  * &#64;XmlRootElement
54  * class Foo {
55  *     &#64;XmlElement
56  *     List<String> data;
57  * }
58  * </pre>
59  *
60  * would produce XML like this:
61  *
62  * <pre>
63  * <foo>
64  *   <data>abc</data>
65  *   <data>def</data>
66  * </foo>
67  * </pre>
68  *
69  * &#64;XmlList annotation, on the other hand, allows multiple values to be
70  * represented as whitespace-separated tokens in a single element. For example,
```

```

71  *
72  * <pre>
73  *   &#64;XmlElement
74  *   class Foo {
75  *       &#64;XmlElement
76  *       &#64;XmlList
77  *       List<String> data;
78  *   }
79  * </pre>
80  *
81  * the above code will produce XML like this:
82  *
83  * <pre>
84  *   <foo>
85  *     <data>abc def</data>
86  *   </foo>
87  * </pre>
88  *
89  * <p>This annotation can be used with the following annotations:
90  *     {@link XmlElement},
91  *     {@link XmlAttribute},
92  *     {@link XmlValue},
93  *     {@link XmlIDREF}.
94  * <ul>
95  *   <li> The use of <tt>@XmlList</tt> with {@link XmlValue} while
96  *     allowed, is redundant since {@link XmlList} maps a
97  *     collection type to a simple schema type that derives by
98  *     list just as {@link XmlValue} would. </li>
99  *
100  *   <li> The use of <tt>@XmlList</tt> with {@link XmlAttribute} while
101  *     allowed, is redundant since {@link XmlList} maps a
102  *     collection type to a simple schema type that derives by
103  *     list just as {@link XmlAttribute} would. </li>
104  * </ul>
105  *
106  * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Sekhar Vajjhala, Sun
107  *   Microsystems, Inc.</li></ul>
108  * @since JAXB2.0
109  */
110 @Retention(RUNTIME) @Target({FIELD,METHOD,PARAMETER})
111 public @interface XmlList {

```

### 3.23 XmlMimeType.java

Secuencia 57: javax/xml/bind/annotation/XmlMimeType.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *

```

```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30 import static java.lang.annotation.RetentionPolicy.RUNTIME;
31 import static java.lang.annotation.ElementType.FIELD;
32 import static java.lang.annotation.ElementType.METHOD;
33 import static java.lang.annotation.ElementType.PARAMETER;
34
35 import javax.xml.transform.Source;
36
37 /**
38  * Associates the MIME type that controls the XML representation of the property.
39  *
40  * <p>
41  * This annotation is used in conjunction with datatypes such as
42  * {@link java.awt.Image} or {@link Source} that are bound to base64-encoded binary in XML.
43  *
44  * <p>
45  * If a property that has this annotation has a sibling property bound to
46  * the xmime:contentType attribute, and if in the instance the property has a value,
47  * the value of the attribute takes precedence and that will control the marshalling.
48  *
49  * @author Kohsuke Kawaguchi
50  * @since JAXB2.0
51  */
52 @Retention(RUNTIME)
53 @Target({FIELD, METHOD, PARAMETER})
54 public @interface XmlMimeType {
55     /**
56      * The textual representation of the MIME type,
57      * such as "image/jpeg" "image/*", "text/xml; charset=iso-8859-1" and so on.
58      */
59     String value();
60 }
```

### 3.24 XmlMixed.java

Secuencia 58: javax/xml/bind/annotation/XmlMixed.java

```
1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30
31 import static java.lang.annotation.RetentionPolicy.RUNTIME;
32 import static java.lang.annotation.ElementType.FIELD;
33 import static java.lang.annotation.ElementType.METHOD;
34
35 import org.w3c.dom.Element;
36 import javax.xml.bind.JAXBElement;
37
38 /**
39  * <p>
40  * Annotate a JavaBean multi-valued property to support mixed content.
41  *
42  * <p>
43  * The usage is subject to the following constraints:
44  * <ul>
45  * <li> can be used with &#64;XmlElementRef, &#64;XmlElementRefs or &#64;XmlAnyElement</li>
46  * </ul>
47  * <p>
48  * The following can be inserted into &#64;XmlMixed annotated multi-valued property
49  * <ul>
50  * <li>XML text information items are added as values of java.lang.String.</li>
```

```

51 * <li>Children element information items are added as instances of
52 * {@link JAXBElement} or instances with a class that is annotated with
53 * {@code XmlRootElement}.

```

```

104 * </letterBody>
105 * </pre>
106 * that can be constructed using following JAXB API calls.
107 * <pre>
108 * LetterBody lb = ObjectFactory.createLetterBody();
109 * JAXBELEMENT<LetterBody> lbe = ObjectFactory.createLetterBody(lb);
110 * List gcl = lb.getContent(); //add mixed content to general content property.
111 * gcl.add("Dear Mr."); // add text information item as a String.
112 *
113 * // add child element information item
114 * gcl.add(ObjectFactory.createLetterBodyName("Robert Smith"));
115 * gcl.add("Your order of "); // add text information item as a String
116 *
117 * // add children element information items
118 * gcl.add(ObjectFactory.
119 *         createLetterBodyQuantity(new BigInteger("1")));
120 * gcl.add(ObjectFactory.createLetterBodyProductName("Baby Monitor"));
121 * gcl.add("shipped from our warehouse"); // add text information item
122 * </pre>
123 *
124 * <p>See "Package Specification" in javax.xml.bind.package javadoc for
125 * additional common information.</p>
126 * @author Kohsuke Kawaguchi
127 * @since JAXB2.0
128 */
129 @Retention(RUNTIME)
130 @Target({FIELD,METHOD})
131 public @interface XmlMixed {
132 }

```

### 3.25 XmlNs.java

Secuencia 59: javax/xml/bind/annotation/XmlNs.java

```

1 /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```



```

21  *
22  *
23  *
24  */
25
26  package javax.xml.bind.annotation;
27
28  import java.lang.annotation.Retention;
29  import static java.lang.annotation.RetentionPolicy.RUNTIME;
30  import java.lang.annotation.Target;
31
32  /**
33   * <p>
34   * Associates a namespace prefix with a XML namespace URI.
35   *
36   * <p><b>Usage</b></p>
37   * <p><tt>@XmlNs</tt> annotation is intended for use from other
38   * program annotations.
39   *
40   * <p>See "Package Specification" in javax.xml.bind.package javadoc for
41   * additional common information.</p>
42   *
43   * <p><b>Example:</b>See <tt>XmlSchema</tt> annotation type for an example.
44   * @author Sekhar Vajjhala, Sun Microsystems, Inc.
45   * @since JAXB2.0
46   */
47
48  @Retention(RUNTIME) @Target({})
49  public @interface XmlNs {
50      /**
51       * Namespace prefix
52       */
53      String prefix();
54
55      /**
56       * Namespace URI
57       */
58      String namespaceURI();
59  }

```

### 3.26 XmlNsForm.java

Secuencia 60: javax/xml/bind/annotation/XmlNsForm.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```

```

11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 /**
29  * Enumeration of XML Schema namespace qualifications.
30  *
31  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
32  * additional common information.</p>
33  *
34  * <p><b>Usage</b></p>
35  * <p>
36  * The namespace qualification values are used in the annotations
37  * defined in this package. The enumeration values are mapped as follows:
38  *
39  * <p>
40  * <table border="1" cellpadding="4" cellspacing="3">
41  *   <tbody>
42  *     <tr>
43  *       <td><b>Enum Value</b></td>
44  *       <td><b>XML Schema Value</b></td>
45  *     </tr>
46  *
47  *     <tr valign="top">
48  *       <td>UNQUALIFIED</td>
49  *       <td>unqualified</td>
50  *     </tr>
51  *     <tr valign="top">
52  *       <td>QUALIFIED</td>
53  *       <td>qualified</td>
54  *     </tr>
55  *     <tr valign="top">
56  *       <td>UNSET</td>
57  *       <td>namespace qualification attribute is absent from the
58  *         XML Schema fragment</td>
59  *     </tr>
60  *   </tbody>
61  * </table>
62  *
63  * @author Sekhar Vajjhala, Sun Microsystems, Inc.

```

```

64  * @since JAXB2.0
65  */
66  public enum XmlNsForm {UNQUALIFIED, QUALIFIED, UNSET}

```

### 3.27 XmlRegistry.java

Secuencia 61: javax/xml/bind/annotation/XmlRegistry.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30 import static java.lang.annotation.ElementType.TYPE;
31 import static java.lang.annotation.RetentionPolicy.RUNTIME;
32
33 /**
34  * Marks a class that has {@link XmlElementDecl}s.
35  *
36  * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Sekhar Vajjhala, Sun
37  *         Microsystems, Inc.</li></ul>
38  * @since JAXB 2.0
39  * @see XmlElementDecl
40  */
41 @Retention(RUNTIME)
42 @Target({TYPE})
43 public @interface XmlRegistry {

```

### 3.28 XmlRootElement.java

Secuencia 62: javax/xml/bind/annotation/XmlRootElement.java

```
1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30
31 import static java.lang.annotation.RetentionPolicy.RUNTIME;
32 import static java.lang.annotation.ElementType.TYPE;
33
34 /**
35  * Maps a class or an enum type to an XML element.
36  *
37  * <p> <b>Usage</b> </p>
38  * <p>
39  * The XmlRootElement annotation can be used with the following program
40  * elements:
41  * <ul>
42  * <li> a top level class </li>
43  * <li> an enum type </li>
44  * </ul>
45  *
46  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
47  * additional common information.</p>
48  *
49  * <p>
50  * When a top level class or an enum type is annotated with the
51  * XmlRootElement annotation, then its value is represented
52  * as XML element in an XML document.
```

```

53 *
54 * <p> This annotation can be used with the following annotations:
55 * {@link XmlType}, {@link XmlEnum}, {@link XmlAccessorType},
56 * {@link XmlAccessorTypeOrder}.
57 * </p>
58
59 * <p>
60 * <b>Example 1: </b> Associate an element with XML Schema type
61 * <pre>
62 *     // Example: Code fragment
63 *     &#64;XmlRootElement
64 *     class Point {
65 *         int x;
66 *         int y;
67 *         Point(int _x,int _y) {x=_x;y=_y;}
68 *     }
69 * </pre>
70 *
71 * <pre>
72 *     //Example: Code fragment corresponding to XML output
73 *     marshal( new Point(3,5), System.out);
74 * </pre>
75 *
76 * <pre>
77 *     &lt;!-- Example: XML output -->
78 *     &lt;point>
79 *         &lt;x> 3 </x>
80 *         &lt;y> 5 </y>
81 *     &lt;/point>
82 * </pre>
83 *
84 * The annotation causes an global element declaration to be produced
85 * in the schema. The global element declaration is associated with
86 * the XML schema type to which the class is mapped.
87 *
88 * <pre>
89 *     &lt;!-- Example: XML schema definition -->
90 *     &lt;xs:element name="point" type="point"/>
91 *     &lt;xs:complexType name="point">
92 *         &lt;xs:sequence>
93 *             &lt;xs:element name="x" type="xs:int"/>
94 *             &lt;xs:element name="y" type="xs:int"/>
95 *         &lt;/xs:sequence>
96 *     &lt;/xs:complexType>
97 * </pre>
98 *
99 * <p>
100 *
101 * <b>Example 2: Orthogonality to type inheritance </b>
102 *
103 * <p>
104 * An element declaration annotated on a type is not inherited by its
105 * derived types. The following example shows this.

```

```

106 * <pre>
107 *     // Example: Code fragment
108 *     &#64;XmlRootElement
109 *     class Point3D extends Point {
110 *         int z;
111 *         Point3D(int _x,int _y,int _z) {super(_x,_y);z=_z;}
112 *     }
113 *
114 *     //Example: Code fragment corresponding to XML output *
115 *     marshal( new Point3D(3,5,0), System.out );
116 *
117 *     &lt;!-- Example: XML output -->
118 *     &lt;!-- The element name is point3D not point -->
119 *     &lt;point3D>
120 *         &lt;x>3&lt;/x>
121 *         &lt;y>5&lt;/y>
122 *         &lt;z>0&lt;/z>
123 *     &lt;/point3D>
124 *
125 *     &lt;!-- Example: XML schema definition -->
126 *     &lt;xs:element name="point3D" type="point3D"/>
127 *     &lt;xs:complexType name="point3D">
128 *         &lt;xs:complexContent>
129 *             &lt;xs:extension base="point">
130 *                 &lt;xs:sequence>
131 *                     &lt;xs:element name="z" type="xs:int"/>
132 *                 &lt;/xs:sequence>
133 *             &lt;/xs:extension>
134 *         &lt;/xs:complexContent>
135 *     &lt;/xs:complexType>
136 * </pre>
137 *
138 * <b>Example 3: </b> Associate a global element with XML Schema type
139 * to which the class is mapped.
140 * <pre>
141 *     //Example: Code fragment
142 *     &#64;XmlRootElement(name="PriceElement")
143 *     public class USPrice {
144 *         &#64;XmlElement
145 *         public java.math.BigDecimal price;
146 *     }
147 *
148 *     &lt;!-- Example: XML schema definition -->
149 *     &lt;xs:element name="PriceElement" type="USPrice"/>
150 *     &lt;xs:complexType name="USPrice">
151 *         &lt;xs:sequence>
152 *             &lt;xs:element name="price" type="xs:decimal"/>
153 *         &lt;/sequence>
154 *     &lt;/xs:complexType>
155 * </pre>
156 *
157 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
158 * @since JAXB2.0

```

```

159  */
160  @Retention(RUNTIME)
161  @Target({TYPE})
162  public @interface XmlRootElement {
163      /**
164       * namespace name of the XML element.
165       * <p>
166       * If the value is "##default", then the XML namespace name is derived
167       * from the package of the class ( {@link XmlSchema} ). If the
168       * package is unnamed, then the XML namespace is the default empty
169       * namespace.
170       */
171      String namespace() default "##default";
172
173      /**
174       * local name of the XML element.
175       * <p>
176       * If the value is "##default", then the name is derived from the
177       * class name.
178       *
179       */
180      String name() default "##default";
181  }
182

```

### 3.29 XmlSchema.java

Secuencia 63: javax/xml/bind/annotation/XmlSchema.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25

```

```

26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30
31 import static java.lang.annotation.ElementType.*;
32 import static java.lang.annotation.RetentionPolicy.*;
33
34 /**
35  * <p> Maps a package name to a XML namespace. </p>
36  *
37  * <h3>Usage</h3>
38  * <p>
39  * The XmlSchema annotation can be used with the following program
40  * elements:
41  * <ul>
42  * <li>package</li>
43  * </ul>
44  *
45  * <p>
46  * This is a package level annotation and follows the recommendations
47  * and restrictions contained in JSR 175, section III, "Annotations".
48  * Thus the usage is subject to the following constraints and
49  * recommendations.
50  * <ul>
51  * <li> There can only be one package declaration as noted in JSR
52  * 175, section III, "Annotations". </li>
53  * <li> JSR 175 recommends package-info.java for package level
54  * annotations. JAXB Providers that follow this recommendation
55  * will allow the package level annotations to be defined in
56  * package-info.java.
57  * </ul>
58  * <p>
59  *
60  * <p><b>Example 1:</b> Customize name of XML namespace to which
61  * package is mapped.</p>
62  *
63  * <pre>
64  *      &#64;javax.xml.bind.annotation.XmlSchema (
65  *          namespace = "http://www.example.com/MYP01"
66  *      )
67  *
68  *      &lt;!-- XML Schema fragment -->
69  *      &lt;schema
70  *          xmlns=...
71  *          xmlns:po=...
72  *          targetNamespace="http://www.example.com/MYP01"
73  *      >
74  *      &lt;!-- prefixes generated by default are implementation
75  *          depedenent -->
76  * </pre>
77  *
78  * <p><b>Example 2:</b> Customize namespace prefix, namespace URI

```



```

79  * mapping</p>
80  *
81  * <pre>
82  *     // Package level annotation
83  *     &#64;javax.xml.bind.annotation.XmlSchema (
84  *         xmlns = {
85  *             &#64;javax.xml.bind.annotation.XmlNs(prefix = "po",
86  *                 namespaceURI="http://www.example.com/myP01"),
87  *
88  *             &#64;javax.xml.bind.annotation.XmlNs(prefix="xs",
89  *                 namespaceURI="http://www.w3.org/2001/XMLSchema")
90  *         )
91  *     )
92  *
93  *     &lt;!-- XML Schema fragment -->
94  *     &lt;schema
95  *         xmlns:xs="http://www.w3.org/2001/XMLSchema"
96  *         xmlns:po="http://www.example.com/P01"
97  *         targetNamespace="http://www.example.com/P01">
98  *
99  * </pre>
100 *
101 * <p><b>Example 3:</b> Customize elementFormDefault</p>
102 * <pre>
103 *     &#64;javax.xml.bind.annotation.XmlSchema (
104 *         elementFormDefault=XmlNsForm.UNQUALIFIED
105 *         ...
106 *     )
107 *
108 *     &lt;!-- XML Schema fragment -->
109 *     &lt;schema
110 *         xmlns="http://www.w3.org/2001/XMLSchema"
111 *         xmlns:po="http://www.example.com/P01"
112 *         elementFormDefault="unqualified">
113 *
114 * </pre>
115 *
116 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
117 * @since JAXB2.0
118 */
119
120 @Retention(RUNTIME) @Target(PACKAGE)
121 public @interface XmlSchema {
122
123     /**
124      * Customize the namespace URI, prefix associations. By default,
125      * the namespace prefixes for a XML namespace are generated by a
126      * JAXB Provider in an implementation dependent way.
127      */
128     XmlNs[] xmlns() default {};
129
130     /**
131      * Name of the XML namespace.

```

```

132     */
133     String namespace() default "";
134
135     /**
136      * Namespace qualification for elements. By default, element
137      * default attribute will be absent from the XML Schema fragment.
138      */
139     XmlNsForm elementFormDefault() default XmlNsForm.UNSET;
140
141     /**
142      * Namespace qualification for attributes. By default,
143      * attributesFormDefault will be absent from the XML Schema fragment.
144      */
145     XmlNsForm attributeFormDefault() default XmlNsForm.UNSET;
146
147     /**
148      * Indicates that this namespace (specified by {@link #namespace()})
149      * has a schema already available externally, available at this location.
150      *
151      * <p>
152      * This instructs the JAXB schema generators to simply refer to
153      * the pointed schema, as opposed to generating components into the schema.
154      * This schema is assumed to match what would be otherwise produced
155      * by the schema generator (same element names, same type names...)
156      *
157      * <p>
158      * This feature is intended to be used when a set of the Java classes
159      * is originally generated from an existing schema, hand-written to
160      * match externally defined schema, or the generated schema is modified
161      * manually.
162      *
163      * <p>
164      * Value could be any absolute URI, like <tt>http://example.org/some.xsd</tt>.
165      * It is also possible to specify the empty string, to indicate
166      * that the schema is externally available but the location is
167      * unspecified (and thus it's the responsibility of the reader of the generate
168      * schema to locate it.) Finally, the default value of this property
169      * <tt>"##generate"</tt> indicates that the schema generator is going
170      * to generate components for this namespace (as it did in JAXB 2.0.)
171      *
172      * <p>
173      * Multiple {@link XmlSchema} annotations on multiple packages are allowed
174      * to govern the same {@link #namespace()}. In such case, all of them
175      * must have the same {@link #location()} values.
176      *
177      *
178      * <h3>Note to implementor</h3>
179      * <p>
180      * More precisely, the value must be either <tt>""</tt>, <tt>"##generate"</tt>, or
181      * <a href="http://www.w3.org/TR/xmlschema-2/#anyURI">
182      * a valid lexical representation of <tt>xs:anyURI</tt></a> that begins
183      * with <tt>&lt;scheme>:</tt>.
184      *
185      *

```

```

185     * <p>
186     * A schema generator is expected to generate a corresponding
187     * <tt><lt;xs:import namespace="..." schemaLocation="...">/></tt> (or
188     * no <tt>schemaLocation</tt> attribute at all if the empty string is specified.)
189     * However, the schema generator is allowed to use a different value in
190     * the <tt>schemaLocation</tt> attribute (including not generating
191     * such attribute), for example so that the user can specify a local
192     * copy of the resource through the command line interface.
193     *
194     * @since JAXB2.1
195     */
196     String location() default NO_LOCATION;
197
198     /**
199     * The default value of the {@link #location()} attribute,
200     * which indicates that the schema generator will generate
201     * components in this namespace.
202     */
203     // the actual value is chosen because ## is not a valid
204     // sequence in xs:anyURI.
205     static final String NO_LOCATION = "##generate";
206 }

```

### 3.30 XmlSchemaType.java

Secuencia 64: javax/xml/bind/annotation/XmlSchemaType.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27

```

```

28 import java.lang.annotation.Retention;
29 import java.lang.annotation.Target;
30
31 import static java.lang.annotation.ElementType.FIELD;
32 import static java.lang.annotation.ElementType.METHOD;
33 import static java.lang.annotation.ElementType.PACKAGE;
34 import static java.lang.annotation.RetentionPolicy.RUNTIME;
35
36 /**
37  * Maps a Java type to a simple schema built-in type.
38  *
39  * <p> <b>Usage</b> </p>
40  * <p>
41  * <tt>@XmlSchemaType</tt> annotation can be used with the following program
42  * elements:
43  * <ul>
44  *   <li> a JavaBean property </li>
45  *   <li> field </li>
46  *   <li> package</li>
47  * </ul>
48  *
49  * <p> <tt>@XmlSchemaType</tt> annotation defined for Java type
50  * applies to all references to the Java type from a property/field.
51  * A <tt>@XmlSchemaType</tt> annotation specified on the
52  * property/field overrides the <tt>@XmlSchemaType</tt> annotation
53  * specified at the package level.
54  *
55  * <p> This annotation can be used with the following annotations:
56  * {@link XmlElement}, {@link XmlAttribute}.
57  * <p>
58  * <b>Example 1: </b> Customize mapping of XMLGregorianCalendar on the
59  * field.
60  *
61  * <pre>
62  *     //Example: Code fragment
63  *     public class USPrice {
64  *         &#64;XmlElement
65  *         &#64;XmlSchemaType(name="date")
66  *         public XMLGregorianCalendar date;
67  *     }
68  *
69  *     &lt;!-- Example: Local XML Schema element -->
70  *     &lt;xs:complexType name="USPrice"/>
71  *     &lt;xs:sequence>
72  *     &lt;xs:element name="date" type="xs:date"/>
73  *     &lt;/sequence>
74  *     &lt;/xs:complexType>
75  * </pre>
76  *
77  * <p> <b> Example 2: </b> Customize mapping of XMLGregorianCalendar at package
78  * level </p>
79  * <pre>
80  *     package foo;

```

```

81  *      &#64;javax.xml.bind.annotation.XmlSchemaType(
82  *          name="date", type=javax.xml.datatype.XMLGregorianCalendar.class)
83  *      }
84  * </pre>
85  *
86  * @since JAXB2.0
87  */
88
89 @Retention(RUNTIME) @Target({FIELD,METHOD,PACKAGE})
90 public @interface XmlSchemaType {
91     String name();
92     String namespace() default "http://www.w3.org/2001/XMLSchema";
93     /**
94      * If this annotation is used at the package level, then value of
95      * the type() must be specified.
96      */
97
98     Class type() default DEFAULT.class;
99
100    /**
101     * Used in {@link XmlSchemaType#type()} to
102     * signal that the type be inferred from the signature
103     * of the property.
104     */
105
106    static final class DEFAULT {}
107
108 }

```

### 3.31 XmlSchemaTypes.java

Secuencia 65: javax/xml/bind/annotation/XmlSchemaTypes.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```

22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import static java.lang.annotation.RetentionPolicy.RUNTIME;
29 import static java.lang.annotation.ElementType.PACKAGE;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.Target;
32
33 /**
34  * <p>
35  * A container for multiple {@link XmlSchemaType} annotations.
36  *
37  * <p> Multiple annotations of the same type are not allowed on a program
38  * element. This annotation therefore serves as a container annotation
39  * for multiple &#64;XmlSchemaType annotations as follows:
40  *
41  * <pre>
42  * &#64;XmlSchemaTypes({ @XmlSchemaType(...), @XmlSchemaType(...) })
43  * </pre>
44  * <p>The <tt>@XmlSchemaTypes</tt> annotation can be used to
45  * define {@link XmlSchemaType} for different types at the
46  * package level.
47  *
48  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
49  * additional common information.</p>
50  *
51  * @author <ul><li>Sekhar Vajjhala, Sun Microsystems, Inc.</li></ul>
52  * @see XmlSchemaType
53  * @since JAXB2.0
54  */
55 @Retention(RUNTIME) @Target({PACKAGE})
56 public @interface XmlSchemaTypes {
57     /**
58      * Collection of {@link XmlSchemaType} annotations
59      */
60     XmlSchemaType[] value();
61 }

```

### 3.32 XmlSeeAlso.java

Secuencia 66: javax/xml/bind/annotation/XmlSeeAlso.java

```

1  /*
2  * Copyright (c) 2006, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```

```
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation;
27
28 import javax.xml.bind.JAXBContext;
29 import java.lang.annotation.ElementType;
30 import java.lang.annotation.Retention;
31 import static java.lang.annotation.RetentionPolicy.RUNTIME;
32 import java.lang.annotation.Target;
33
34 /**
35  * Instructs JAXB to also bind other classes when binding this class.
36  *
37  * <p>
38  * Java makes it impractical/impossible to list all sub-classes of
39  * a given class. This often gets in a way of JAXB users, as it JAXB
40  * cannot automatically list up the classes that need to be known
41  * to {@link JAXBContext}.
42  *
43  * <p>
44  * For example, with the following class definitions:
45  *
46  * <pre>
47  * class Animal {}
48  * class Dog extends Animal {}
49  * class Cat extends Animal {}
50  * </pre>
51  *
52  * <p>
53  * The user would be required to create {@link JAXBContext} as
54  * <tt>JAXBContext.newInstance(Dog.class,Cat.class)</tt>
55  * (<tt>Animal</tt> will be automatically picked up since <tt>Dog</tt>
56  * and <tt>Cat</tt> refers to it.)
57  *
58  * <p>
59  * {@link XmlSeeAlso} annotation would allow you to write:
60  * <pre>
61  * &#64;XmlSeeAlso({Dog.class,Cat.class})
62  * class Animal {}
```

```

63  * class Dog extends Animal {}
64  * class Cat extends Animal {}
65  * </pre>
66  *
67  * <p>
68  * This would allow you to do <tt>JAXBContext.newInstance(Animal.class)</tt>.
69  * By the help of this annotation, JAXB implementations will be able to
70  * correctly bind <tt>Dog</tt> and <tt>Cat</tt>.
71  *
72  * @author Kohsuke Kawaguchi
73  * @since JAXB2.1
74  */
75 @Target({ElementType.TYPE})
76 @Retention(RUNTIME)
77 public @interface XmlSeeAlso {
78     Class[] value();
79 }

```

### 3.33 XmlTransient.java

Secuencia 67: javax/xml/bind/annotation/XmlTransient.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Target;
29 import java.lang.annotation.Retention;
30 import static java.lang.annotation.ElementType.*;
31 import static java.lang.annotation.RetentionPolicy.*;
32

```



```

33 /**
34  * <p>
35  * Prevents the mapping of a JavaBean property/type to XML representation.
36  * <p>
37  * The <tt>@XmlTransient</tt> annotation is useful for resolving name
38  * collisions between a JavaBean property name and a field name or
39  * preventing the mapping of a field/property. A name collision can
40  * occur when the decapitalized JavaBean property name and a field
41  * name are the same. If the JavaBean property refers to the field,
42  * then the name collision can be resolved by preventing the
43  * mapping of either the field or the JavaBean property using the
44  * <tt>@XmlTransient</tt> annotation.
45  *
46  * <p>
47  * When placed on a class, it indicates that the class shouldn't be mapped
48  * to XML by itself. Properties on such class will be mapped to XML along
49  * with its derived classes, as if the class is inlined.
50  *
51  * <p><b>Usage</b></p>
52  * <p> The <tt>@XmlTransient</tt> annotation can be used with the following
53  *   program elements:
54  * <ul>
55  *   <li> a JavaBean property </li>
56  *   <li> field </li>
57  *   <li> class </li>
58  * </ul>
59  *
60  * <p><tt>@XmlTransient</tt> is mutually exclusive with all other
61  * JAXB defined annotations. </p>
62  *
63  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
64  * additional common information.</p>
65  *
66  * <p><b>Example:</b> Resolve name collision between JavaBean property and
67  *   field name </p>
68  *
69  * <pre>
70  * // Example: Code fragment
71  * public class USAddress {
72  *
73  *     // The field name "name" collides with the property name
74  *     // obtained by bean decapitalization of getName() below
75  *     &#64;XmlTransient public String name;
76  *
77  *     String getName() {..};
78  *     String setName() {..};
79  * }
80  *
81  *
82  * &lt;!-- Example: XML Schema fragment -->
83  * &lt;xs:complexType name="USAddress">
84  *   &lt;xs:sequence>
85  *     &lt;xs:element name="name" type="xs:string"/>

```

```

86 *      </xs:sequence>
87 *      </xs:complexType>
88 * </pre>
89 *
90 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
91 * @since JAXB2.0
92 */
93
94 @Retention(RUNTIME) @Target({FIELD, METHOD, TYPE})
95 public @interface XmlTransient {}

```

### 3.34 XmlType.java

Secuencia 68: javax/xml/bind/annotation/XmlType.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import static java.lang.annotation.ElementType.TYPE;
29 import java.lang.annotation.Retention;
30 import static java.lang.annotation.RetentionPolicy.RUNTIME;
31 import java.lang.annotation.Target;
32
33 /**
34 * <p>
35 * Maps a class or an enum type to a XML Schema type.
36 *
37 * <p><b>Usage</b></p>
38 * <p> The <tt>@XmlType</tt> annotation can be used with the following program
39 * elements:

```

```

40 * <ul>
41 *   <li> a top level class </li>
42 *   <li> an enum type </li>
43 * </ul>
44 *
45 * <p>See "Package Specification" in javax.xml.bind.package javadoc for
46 * additional common information.</p>
47 *
48 * <h3> Mapping a Class </h3>
49 * <p>
50 * A class maps to a XML Schema type. A class is a data container for
51 * values represented by properties and fields. A schema type is a
52 * data container for values represented by schema components within a
53 * schema type's content model (e.g. model groups, attributes etc).
54 * <p> To be mapped, a class must either have a public no-arg
55 * constructor or a static no-arg factory method. The static factory
56 * method can be specified in <tt>factoryMethod()</tt> and
57 * <tt>factoryClass()</tt> annotation elements. The static factory
58 * method or the no-arg constructor is used during unmarshalling to
59 * create an instance of this class. If both are present, the static
60 * factory method overrides the no-arg constructor.
61 * <p>
62 * A class maps to either a XML Schema complex type or a XML Schema simple
63 * type. The XML Schema type is derived based on the
64 * mapping of JavaBean properties and fields contained within the
65 * class. The schema type to which the class is mapped can either be
66 * named or anonymous. A class can be mapped to an anonymous schema
67 * type by annotating the class with <tt>XmlType(name="")</tt>.
68 * <p>
69 * Either a global element, local element or a local attribute can be
70 * associated with an anonymous type as follows:
71 * <ul>
72 *   <li><b>global element: </b> A global element of an anonymous
73 *     type can be derived by annotating the class with @{@link
74 *     XmlRootElement}. See Example 3 below. </li>
75 *
76 *   <li><b>local element: </b> A JavaBean property that references
77 *     a class annotated with @XmlType(name="") and is mapped to the
78 *     element associated with the anonymous type. See Example 4
79 *     below.</li>
80 *
81 *   <li><b>attribute: </b> A JavaBean property that references
82 *     a class annotated with @XmlType(name="") and is mapped to the
83 *     attribute associated with the anonymous type. See Example 5 below. </li>
84 * </ul>
85 * <b> Mapping to XML Schema Complex Type </b>
86 * <ul>
87 *   <li>If class is annotated with <tt>XmlType(name="")</tt>, it
88 *   is mapped to an anonymous type otherwise, the class name maps
89 *   to a complex type name. The <tt>XmlName()</tt> annotation element
90 *   can be used to customize the name.</li>
91 *
92 *   <li> Properties and fields that are mapped to elements are mapped to a

```

```

93 * content model within a complex type. The annotation element
94 * <tt>propOrder()</tt> can be used to customize the content model to be
95 * <tt>xs:all</tt> or <tt>xs:sequence</tt>. It is used for specifying
96 * the order of XML elements in <tt>xs:sequence</tt>. </li>
97 *
98 * <li> Properties and fields can be mapped to attributes within the
99 * complex type. </li>
100 *
101 * <li> The targetnamespace of the XML Schema type can be customized
102 * using the annotation element <tt>namespace()</tt>. </li>
103 * </ul>
104 *
105 * <p>
106 * <b> Mapping class to XML Schema simple type </b>
107 * <p>
108 * A class can be mapped to a XML Schema simple type using the
109 * <tt>@XmlValue</tt> annotation. For additional details and examples,
110 * see @<link XmlValue> annotation type.
111 * <p>
112 * The following table shows the mapping of the class to a XML Schema
113 * complex type or simple type. The notational symbols used in the table are:
114 * <ul>
115 * <li> -> : represents a mapping </li>
116 * <li> [x]+ : one or more occurrences of x </li>
117 * <li> [ <tt>@XmlValue</tt> property ]: JavaBean property annotated with
118 * <tt>@XmlValue</tt></li>
119 * <li> X : don't care
120 * </ul>
121 * <blockquote>
122 * <table border="1" cellpadding="4" cellspacing="3">
123 * <tbody>
124 * <tr>
125 * <td><b>Target</b></td>
126 * <td><b>propOrder</b></td>
127 * <td><b>ClassBody</b></td>
128 * <td><b>ComplexType</b></td>
129 * <td><b>SimpleType</b></td>
130 * </tr>
131 *
132 * <tr valign="top">
133 * <td>Class</td>
134 * <td>{}</td>
135 * <td>[property]+ -> elements</td>
136 * <td>complexContent<br>xs:all</td>
137 * <td></td>
138 * </tr>
139 *
140 * <tr valign="top">
141 * <td>Class</td>
142 * <td>non empty</td>
143 * <td>[property]+ -> elements</td>
144 * <td>complexContent<br>xs:sequence</td>
145 * <td></td>

```

```

146 *      </tr>
147 *
148 *      <tr valign="top">
149 *          <td>Class</td>
150 *          <td>X</td>
151 *          <td>no property -> element</td>
152 *          <td>complexContent<br>empty sequence</td>
153 *          <td> </td>
154 *      </tr>
155 *
156 *      <tr valign="top">
157 *          <td>Class</td>
158 *          <td>X</td>
159 *          <td>1 [ <tt>@XmlValue</tt> property] && <br> [property]+
160 *              ->attributes</td>
161 *          <td>simplecontent</td>
162 *          <td> </td>
163 *      </tr>
164 *
165 *      <tr valign="top">
166 *          <td>Class</td>
167 *          <td>X</td>
168 *          <td>1 [ <tt>@XmlValue</tt> property ]&& <br> no properties
169 *              -> attribute</td>
170 *          <td> </td>
171 *          <td>simplotype</td>
172 *          <td> </td>
173 *      </tr>
174 *  </tbody>
175 * </table>
176 * </blockquote>
177 *
178 * <h3> Mapping an enum type </h3>
179 *
180 * An enum type maps to a XML schema simple type with enumeration
181 * facets. The following annotation elements are ignored since they
182 * are not meaningful: <tt>propOrder()</tt> , <tt>factoryMethod()</tt> ,
183 * <tt>factoryClass()</tt> .
184 *
185 * <h3> Usage with other annotations </h3>
186 * <p> This annotation can be used with the following annotations:
187 * {<link XmlRootElement>}, {<link XmlAccessorType>}, {<link XmlAccessorType>},
188 * {<link XmlEnum>}. However, {<link
189 * XmlAccessorType> and {<link XmlAccessorType> are ignored when this
190 * annotation is used on an enum type.
191 *
192 * <p> <b> Example 1: </b> Map a class to a complex type with
193 *   xs:sequence with a customized ordering of JavaBean properties.
194 * </p>
195 *
196 * <pre>
197 *   &#64;XmlType(propOrder={"street", "city" , "state", "zip", "name" })
198 *   public class USAddress {

```

```

199 *     String getName() {...};
200 *     void setName(String) {...};
201 *
202 *     String getStreet() {...};
203 *     void setStreet(String) {...};
204 *
205 *     String getCity() {...};
206 *     void setCity(String) {...};
207 *
208 *     String getState() {...};
209 *     void setState(String) {...};
210 *
211 *     java.math.BigDecimal getZip() {...};
212 *     void setZip(java.math.BigDecimal) {...};
213 * }
214 *
215 * <!-- XML Schema mapping for USAddress -->
216 * <xs:complexType name="USAddress">
217 *     <xs:sequence>
218 *         <xs:element name="street" type="xs:string"/>
219 *         <xs:element name="city" type="xs:string"/>
220 *         <xs:element name="state" type="xs:string"/>
221 *         <xs:element name="zip" type="xs:decimal"/>
222 *         <xs:element name="name" type="xs:string"/>
223 *     </xs:sequence>
224 * </xs:complexType>
225 * </pre>
226 * <p> <b> Example 2: </b> Map a class to a complex type with
227 *     xs:all </p>
228 * <pre>
229 * &#64;XmlType(propOrder={})
230 * public class USAddress { ...}
231 *
232 * <!-- XML Schema mapping for USAddress -->
233 * <xs:complexType name="USAddress">
234 *     <xs:all>
235 *         <xs:element name="name" type="xs:string"/>
236 *         <xs:element name="street" type="xs:string"/>
237 *         <xs:element name="city" type="xs:string"/>
238 *         <xs:element name="state" type="xs:string"/>
239 *         <xs:element name="zip" type="xs:decimal"/>
240 *     </xs:sequence>
241 * </xs:complexType>
242 * </pre>
243 * <p> <b> Example 3: </b> Map a class to a global element with an
244 *     anonymous type.
245 * </p>
246 * <pre>
247 * &#64;XmlRootElement
248 * &#64;XmlType(name="")
249 * public class USAddress { ...}
250 *
251 * <!-- XML Schema mapping for USAddress -->

```

```

252 * <!-->
253 * <!-->
254 * <!-->
255 * <!-->
256 * <!-->
257 * <!-->
258 * <!-->
259 * <!-->
260 * <!-->
261 * <!-->
262 * <!-->
263 * </pre>
264 *
265 * <p> <b> Example 4: </b> Map a property to a local element with
266 * anonymous type.
267 * <pre>
268 * //Example: Code fragment
269 * public class Invoice {
270 *     USAddress addr;
271 *     ...
272 * }
273 *
274 * <pre>
275 * public class USAddress { ... }
276 * }
277 *
278 * <!-- XML Schema mapping for USAddress -->
279 * <!-->
280 * <!-->
281 * <!-->
282 * <!-->
283 * <!-->
284 * <!-->
285 * <!-->
286 * <!-->
287 * <!-->
288 * <!-->
289 * ...
290 * <!-->
291 * <!-->
292 * </pre>
293 *
294 * <p> <b> Example 5: </b> Map a property to an attribute with
295 * anonymous type.
296 *
297 * <pre>
298 *
299 * //Example: Code fragment
300 * public class Item {
301 *     public String name;
302 *     <!-->
303 *     public USPrice price;
304 * }

```

```

305 *
306 * // map class to anonymous simple type.
307 * &#64;XmlType(name="")
308 * public class USPrice {
309 *     &#64;XmlValue
310 *     public java.math.BigDecimal price;
311 * }
312 *
313 * &lt;!-- Example: XML Schema fragment -->
314 * &lt;xs:complexType name="Item">
315 *     &lt;xs:sequence>
316 *         &lt;xs:element name="name" type="xs:string"/>
317 *         &lt;xs:attribute name="price">
318 *             &lt;xs:simpleType>
319 *                 &lt;xs:restriction base="xs:decimal"/>
320 *             &lt;/xs:simpleType>
321 *         &lt;/xs:attribute>
322 *     &lt;/xs:sequence>
323 * &lt;/xs:complexType>
324 * </pre>
325 *
326 * <p> <b> Example 6: </b> Define a factoryClass and factoryMethod
327 *
328 * <pre>
329 *     &#64;XmlType(name="USAddressType", factoryClass=USAddressFactory.class,
330 *     factoryMethod="getUSAddress")
331 *     public class USAddress {
332 *
333 *         private String city;
334 *         private String name;
335 *         private String state;
336 *         private String street;
337 *         private int    zip;
338 *
339 *         public USAddress(String name, String street, String city,
340 *         String state, int zip) {
341 *             this.name = name;
342 *             this.street = street;
343 *             this.city = city;
344 *             this.state = state;
345 *             this.zip = zip;
346 *         }
347 *     }
348 *
349 *     public class USAddressFactory {
350 *         public static USAddress getUSAddress(){
351 *             return new USAddress("Mark Baker", "23 Elm St",
352 *             "Dayton", "OH", 90952);
353 *         }
354 *     }
355 * </pre>
356 *
357 * <p> <b> Example 7: </b> Define factoryMethod and use the default factoryClass

```



```

358 *
359 * <pre>
360 *      &#64;XmlType(name="USAddressType", factoryMethod="getNewInstance")
361 *      public class USAddress {
362 *
363 *          private String city;
364 *          private String name;
365 *          private String state;
366 *          private String street;
367 *          private int    zip;
368 *
369 *          private USAddress() {}
370 *
371 *          public static USAddress getNewInstance(){
372 *              return new USAddress();
373 *          }
374 *      }
375 * </pre>
376 *
377 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
378 * @see XmlElement
379 * @see XmlAttribute
380 * @see XmlValue
381 * @see XmlSchema
382 * @since JAXB2.0
383 */
384
385 @Retention(RUNTIME) @Target({TYPE})
386 public @interface XmlType {
387     /**
388      * Name of the XML Schema type which the class is mapped.
389      */
390     String name() default "##default" ;
391
392     /**
393      * Specifies the order for XML Schema elements when class is
394      * mapped to a XML Schema complex type.
395      *
396      * <p> Refer to the table for how the propOrder affects the
397      * mapping of class </p>
398      *
399      * <p> The propOrder is a list of names of JavaBean properties in
400      * the class. Each name in the list is the name of a Java
401      * identifier of the JavaBean property. The order in which
402      * JavaBean properties are listed is the order of XML Schema
403      * elements to which the JavaBean properties are mapped. </p>
404      * <p> All of the JavaBean properties being mapped to XML Schema elements
405      * must be listed.
406      * <p> A JavaBean property or field listed in propOrder must not
407      * be transient or annotated with <tt>@XmlTransient</tt>.
408      * <p> The default ordering of JavaBean properties is determined
409      * by @link XmlAccessorType.
410      */

```

```

411 String[] propOrder() default {"";
412
413 /**
414  * Name of the target namespace of the XML Schema type. By
415  * default, this is the target namespace to which the package
416  * containing the class is mapped.
417  */
418 String namespace() default "##default" ;
419
420 /**
421  * Class containing a no-arg factory method for creating an
422  * instance of this class. The default is this class.
423  *
424  * <p>If <tt>factoryClass</tt> is DEFAULT.class and
425  * <tt>factoryMethod</tt> is "", then there is no static factory
426  * method.
427  *
428  * <p>If <tt>factoryClass</tt> is DEFAULT.class and
429  * <tt>factoryMethod</tt> is not "", then
430  * <tt>factoryMethod</tt> is the name of a static factory method
431  * in this class.
432  *
433  * <p>If <tt>factoryClass</tt> is not DEFAULT.class, then
434  * <tt>factoryMethod</tt> must not be "" and must be the name of
435  * a static factory method specified in <tt>factoryClass</tt>.
436  */
437 Class factoryClass() default DEFAULT.class;
438
439 /**
440  * Used in {@link XmlType#factoryClass()} to
441  * signal that either factory method is not used or
442  * that it's in the class with this {@link XmlType} itself.
443  */
444 static final class DEFAULT {}
445
446 /**
447  * Name of a no-arg factory method in the class specified in
448  * <tt>factoryClass</tt> factoryClass().
449  *
450  */
451 String factoryMethod() default "";
452 }

```

### 3.35 XmlValue.java

Secuencia 69: javax/xml/bind/annotation/XmlValue.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *

```

```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation;
27
28 import java.lang.annotation.Target;
29 import java.lang.annotation.Retention;
30 import static java.lang.annotation.ElementType.*;
31 import static java.lang.annotation.RetentionPolicy.*;
32
33 /**
34  * <p>
35  * Enables mapping a class to a XML Schema complex type with a
36  * simpleContent or a XML Schema simple type.
37  * </p>
38  *
39  * <p>
40  * <b> Usage: </b>
41  * <p>
42  * The <tt>@XmlValue</tt> annotation can be used with the following program
43  * elements:
44  * <ul>
45  * <li> a JavaBean property.</li>
46  * <li> non static, non transient field.</li>
47  * </ul>
48  *
49  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
50  * additional common information.</p>
51  *
52  * The usage is subject to the following usage constraints:
53  * <ul>
54  * <li>At most one field or property can be annotated with the
55  * <tt>@XmlValue</tt> annotation. </li>
56  *
57  * <li><tt>@XmlValue</tt> can be used with the following
58  * annotations: {@link XmlList}. However this is redundant since
59  * {@link XmlList} maps a type to a simple schema type that derives by
60  * list just as {@link XmlValue} would. </li>
```

```

61 *
62 * <li>If the type of the field or property is a collection type,
63 *     then the collection item type must map to a simple schema
64 *     type. </li>
65 *
66 * <li>If the type of the field or property is not a collection
67 *     type, then the type must map to a XML Schema simple type. </li>
68 *
69 * </ul>
70 * </p>
71 * <p>
72 * If the annotated JavaBean property is the sole class member being
73 * mapped to XML Schema construct, then the class is mapped to a
74 * simple type.
75 *
76 * If there are additional JavaBean properties (other than the
77 * JavaBean property annotated with <tt>@XmlValue</tt> annotation)
78 * that are mapped to XML attributes, then the class is mapped to a
79 * complex type with simpleContent.
80 * </p>
81 *
82 * <p> <b> Example 1: </b> Map a class to XML Schema simpleType</p>
83 *
84 * <pre>
85 *
86 *     // Example 1: Code fragment
87 *     public class USPrice {
88 *         &#64;XmlValue
89 *         public java.math.BigDecimal price;
90 *     }
91 *
92 *     &lt;!-- Example 1: XML Schema fragment -->
93 *     &lt;xs:simpleType name="USPrice">
94 *         &lt;xs:restriction base="xs:decimal"/>
95 *     &lt;/xs:simpleType>
96 *
97 * </pre>
98 *
99 * <p><b> Example 2: </b> Map a class to XML Schema complexType with
100 *     with simpleContent.</p>
101 *
102 * <pre>
103 *
104 *     // Example 2: Code fragment
105 *     public class InternationalPrice {
106 *         &#64;XmlValue
107 *         public java.math.BigDecimal price;
108 *
109 *         &#64;XmlAttribute
110 *         public String currency;
111 *     }
112 *
113 *     &lt;!-- Example 2: XML Schema fragment -->

```

```

114 * <xs:complexType name="InternationalPrice">
115 *   <xs:simpleContent>
116 *     <xs:extension base="xs:decimal">
117 *       <xs:attribute name="currency" type="xs:string"/>
118 *     </xs:extension>
119 *   </xs:simpleContent>
120 * </xs:complexType>
121 *
122 * </pre>
123 * </p>
124 *
125 * @author Sekhar Vajjhala, Sun Microsystems, Inc.
126 * @see XmlType
127 * @since JAXB2.0
128 */
129
130 @Retention(RUNTIME) @Target({FIELD, METHOD})
131 public @interface XmlValue {}

```

## 4 Xml Bind Annotation Adapters (javax.xml.bind.annotation.adapters package)

### 4.1 CollapsedStringAdapter.java

Secuencia 70: javax/xml/bind/annotation/adapters/CollapsedStringAdapter.java

```

1 /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation.adapters;
27

```

```

28
29
30 /**
31  * Built-in {@link XmlAdapter} to handle <tt>xs:token</tt> and its derived types.
32  *
33  * <p>
34  * This adapter removes leading and trailing whitespaces, then truncate any
35  * sequence of tab, CR, LF, and SP by a single whitespace character ' '.
36  *
37  * @author Kohsuke Kawaguchi
38  * @since JAXB 2.0
39  */
40 public class CollapsedStringAdapter extends XmlAdapter<String,String> {
41     /**
42      * Removes leading and trailing whitespaces of the string
43      * given as the parameter, then truncate any
44      * sequence of tab, CR, LF, and SP by a single whitespace character ' '.
45      */
46     public String unmarshal(String text) {
47         if(text==null) return null;          // be defensive
48
49         int len = text.length();
50
51         // most of the texts are already in the collapsed form.
52         // so look for the first whitespace in the hope that we will
53         // never see it.
54         int s=0;
55         while(s<len) {
56             if(isWhiteSpace(text.charAt(s)))
57                 break;
58             s++;
59         }
60         if(s==len)
61             // the input happens to be already collapsed.
62             return text;
63
64         // we now know that the input contains spaces.
65         // let's sit down and do the collapsing normally.
66
67         StringBuilder result = new StringBuilder(len /*allocate enough size to avoid re-allocation
68             */ );
69
70         if(s!=0) {
71             for( int i=0; i<s; i++ )
72                 result.append(text.charAt(i));
73             result.append(' ');
74         }
75
76         boolean inStripMode = true;
77         for (int i = s+1; i < len; i++) {
78             char ch = text.charAt(i);
79             boolean b = isWhiteSpace(ch);
80             if (inStripMode && b)

```

```

80         continue; // skip this character
81
82         inStripMode = b;
83         if (inStripMode)
84             result.append(' ');
85         else
86             result.append(ch);
87     }
88
89     // remove trailing whitespaces
90     len = result.length();
91     if (len > 0 && result.charAt(len - 1) == ' ')
92         result.setLength(len - 1);
93     // whitespaces are already collapsed,
94     // so all we have to do is to remove the last one character
95     // if it's a whitespace.
96
97     return result.toString();
98 }
99
100 /**
101  * No-op.
102  *
103  * Just return the same string given as the parameter.
104  */
105 public String marshal(String s) {
106     return s;
107 }
108
109
110 /** returns true if the specified char is a white space character. */
111 protected static boolean isWhiteSpace(char ch) {
112     // most of the characters are non-control characters.
113     // so check that first to quickly return false for most of the cases.
114     if( ch>0x20 ) return false;
115
116     // other than we have to do four comparisons.
117     return ch == 0x9 || ch == 0xA || ch == 0xD || ch == 0x20;
118 }
119 }

```

## 4.2 HexBinaryAdapter.java

Secuencia 71: javax/xml/bind/annotation/adapters/HexBinaryAdapter.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```

```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation.adapters;
27
28 import javax.xml.bind.DatatypeConverter;
29
30 /**
31  * {@link XmlAdapter} for <tt>xs:hexBinary</tt>.
32  *
33  * <p>
34  * This {@link XmlAdapter} binds <tt>byte[]</tt> to the hexBinary representation in XML.
35  *
36  * @author Kohsuke Kawaguchi
37  * @since JAXB 2.0
38  */
39 public final class HexBinaryAdapter extends XmlAdapter<String,byte[]> {
40     public byte[] unmarshal(String s) {
41         if(s==null) return null;
42         return DatatypeConverter.parseHexBinary(s);
43     }
44
45     public String marshal(byte[] bytes) {
46         if(bytes==null) return null;
47         return DatatypeConverter.printHexBinary(bytes);
48     }
49 }

```

### 4.3 NormalizedStringAdapter.java

Secuencia 72: javax/xml/bind/annotation/adapters/NormalizedStringAdapter.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```



```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.bind.annotation.adapters;
27
28
29
30  /**
31   * {@link XmlAdapter} to handle <tt>xs:normalizedString</tt>.
32   *
33   * <p>
34   * Replaces any tab, CR, and LF by a whitespace character ' ',
35   * as specified in <a href="http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace">the whitespace facet '
36   * replace'</a>
37   *
38   * @author Kohsuke Kawaguchi, Martin Grebac
39   * @since JAXB 2.0
40   */
41  public final class NormalizedStringAdapter extends XmlAdapter<String,String> {
42      /**
43       * Replace any tab, CR, and LF by a whitespace character ' ',
44       * as specified in <a href="http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace">the whitespace
45       * facet 'replace'</a>
46       */
47      public String unmarshal(String text) {
48          if(text==null)      return null;    // be defensive
49
50          int i=text.length()-1;
51
52          // look for the first whitespace char.
53          while( i>=0 && !isWhiteSpaceExceptSpace(text.charAt(i)) )
54              i--;
55
56          if( i<0 )
57              // no such whitespace. replace(text)==text.
58              return text;
59
60          // we now know that we need to modify the text.
61          // allocate a char array to do it.
62          char[] buf = text.toCharArray();

```

```

61         buf[i--] = ' ';
62         for( ; i>=0; i-- )
63             if( isWhiteSpaceExceptSpace(buf[i]))
64                 buf[i] = ' ';
65
66         return new String(buf);
67     }
68
69     /**
70     * No-op.
71     *
72     * Just return the same string given as the parameter.
73     */
74     public String marshal(String s) {
75         return s;
76     }
77
78
79     /**
80     * Returns true if the specified char is a white space character
81     * but not 0x20.
82     */
83     protected static boolean isWhiteSpaceExceptSpace(char ch) {
84         // most of the characters are non-control characters.
85         // so check that first to quickly return false for most of the cases.
86         if( ch>=0x20 ) return false;
87
88         // other than we have to do four comparisons.
89         return ch == 0x9 || ch == 0xA || ch == 0xD;
90     }
91 }
92 }

```

#### 4.4 XmlAdapter.java

Secuencia 73: javax/xml/bind/annotation/adapters/XmlAdapter.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```

18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation.adapters;
27
28 /**
29  * Adapts a Java type for custom marshaling.
30  *
31  * <p> <b> Usage: </b> </p>
32  *
33  * <p>
34  * Some Java types do not map naturally to a XML representation, for
35  * example <tt>HashMap</tt> or other non JavaBean classes. Conversely,
36  * a XML representation may map to a Java type but an application may
37  * choose to access the XML representation using another Java
38  * type. For example, the schema to Java binding rules bind
39  * xs:DateTime by default to XmlGregorianCalendar. But an application
40  * may desire to bind xs:DateTime to a custom type,
41  * MyXmlGregorianCalendar, for example. In both cases, there is a
42  * mismatch between <i> bound type </i>, used by an application to
43  * access XML content and the <i> value type</i>, that is mapped to an
44  * XML representation.
45  *
46  * <p>
47  * This abstract class defines methods for adapting a bound type to a value
48  * type or vice versa. The methods are invoked by the JAXB binding
49  * framework during marshaling and unmarshalling:
50  *
51  * <ul>
52  * <li> <b> XmlAdapter.marshal(...): </b> During marshalling, JAXB
53  * binding framework invokes XmlAdapter.marshal(..) to adapt a
54  * bound type to value type, which is then marshaled to XML
55  * representation. </li>
56  *
57  * <li> <b> XmlAdapter.unmarshal(...): </b> During unmarshalling,
58  * JAXB binding framework first unmarshals XML representation
59  * to a value type and then invokes XmlAdapter.unmarshal(..) to
60  * adapt the value type to a bound type. </li>
61  * </ul>
62  *
63  * Writing an adapter therefore involves the following steps:
64  *
65  * <ul>
66  * <li> Write an adapter that implements this abstract class. </li>
67  * <li> Install the adapter using the annotation {@link
68  * XmlJavaTypeAdapter} </li>
69  * </ul>
70  *

```

```

71 * <p><b>Example:</b> Customized mapping of <tt>HashMap</tt></p>
72 * <p> The following example illustrates the use of
73 * <tt>XmlAdapter</tt> and <tt>XmlJavaTypeAdapter</tt> to
74 * customize the mapping of a <tt>HashMap</tt>.
75 *
76 * <p> <b> Step 1: </b> Determine the desired XML representation for HashMap.
77 *
78 * <pre>
79 *     <tt>hashmap>
80 *         <tt>entry key="id123">this is a value</tt></entry>
81 *         <tt>entry key="id312">this is another value</tt></entry>
82 *         ...
83 *     </tt></hashmap>
84 * </pre>
85 *
86 * <p> <b> Step 2: </b> Determine the schema definition that the
87 * desired XML representation shown above should follow.
88 *
89 * <pre>
90 *
91 *     <tt>xs:complexType name="myHashMapType">
92 *         <tt>xs:sequence>
93 *             <tt>xs:element name="entry" type="myHashMapEntryType"
94 *                 minOccurs = "0" maxOccurs="unbounded"/>
95 *         </tt></xs:sequence>
96 *     </tt></xs:complexType>
97 *
98 *     <tt>xs:complexType name="myHashMapEntryType">
99 *         <tt>xs:simpleContent>
100 *             <tt>xs:extension base="xs:string">
101 *                 <tt>xs:attribute name="key" type="xs:int"/>
102 *             </tt></xs:extension>
103 *         </tt></xs:simpleContent>
104 *     </tt></xs:complexType>
105 *
106 * </pre>
107 *
108 * <p> <b> Step 3: </b> Write value types that can generate the above
109 * schema definition.
110 *
111 * <pre>
112 *     public class MyHashMapType {
113 *         List<MyHashMapEntryType> entry;
114 *     }
115 *
116 *     public class MyHashMapEntryType {
117 *         XmlAttribute
118 *         public Integer key;
119 *
120 *         XmlValue
121 *         public String value;
122 *     }
123 * </pre>

```

```

124 *
125 * <p> <b> Step 4: </b> Write the adapter that adapts the value type,
126 * MyHashMapType to a bound type, HashMap, used by the application.
127 *
128 * <pre>
129 *     public final class MyHashMapAdapter extends
130 *         XmlAdapter<MyHashMapType,HashMap> { ... }
131 *
132 * </pre>
133 *
134 * <p> <b> Step 5: </b> Use the adapter.
135 *
136 * <pre>
137 *     public class Foo {
138 *         &#64;XmlJavaTypeAdapter(MyHashMapAdapter.class)
139 *         HashMap hashmap;
140 *         ...
141 *     }
142 * </pre>
143 *
144 * The above code fragment will map to the following schema:
145 *
146 * <pre>
147 *     <xs:complexType name="Foo">
148 *         <xs:sequence>
149 *             <xs:element name="hashmap" type="myHashMapType"
150 *             </xs:sequence>
151 *         </xs:complexType>
152 * </pre>
153 *
154 * @param <BoundType>
155 *     The type that JAXB doesn't know how to handle. An adapter is written
156 *     to allow this type to be used as an in-memory representation through
157 *     the <tt>ValueType</tt>.
158 * @param <ValueType>
159 *     The type that JAXB knows how to handle out of the box.
160 *
161 * @author <ul><li>Sekhar Vajjhala, Sun Microsystems Inc.</li> <li>Kohsuke Kawaguchi, Sun
162 *     Microsystems Inc.</li></ul>
163 * @see XmlJavaTypeAdapter
164 * @since JAXB 2.0
165 */
166 public abstract class XmlAdapter<ValueType,BoundType> {
167     /**
168     * Do-nothing constructor for the derived classes.
169     */
170     protected XmlAdapter() {}
171
172     /**
173     * Convert a value type to a bound type.
174     *
175     * @param v

```

```

176     *      The value to be converted. Can be null.
177     * @throws Exception
178     *      if there's an error during the conversion. The caller is responsible for
179     *      reporting the error to the user through {@link javax.xml.bind.ValidationEventHandler}.
180     */
181     public abstract BoundType unmarshal(ValueType v) throws Exception;
182
183     /**
184     * Convert a bound type to a value type.
185     *
186     * @param v
187     *      The value to be converted. Can be null.
188     * @throws Exception
189     *      if there's an error during the conversion. The caller is responsible for
190     *      reporting the error to the user through {@link javax.xml.bind.ValidationEventHandler}.
191     */
192     public abstract ValueType marshal(BoundType v) throws Exception;
193 }

```

#### 4.5 XmlJavaTypeAdapter.java

Secuencia 74: javax/xml/bind/annotation/adapters/XmlJavaTypeAdapter.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.annotation.adapters;
27
28 import javax.xml.bind.annotation.XmlAnyElement;
29 import javax.xml.bind.annotation.XmlElementRefs;
30 import javax.xml.bind.annotation.XmlElement;
31 import javax.xml.bind.annotation.XmlSchemaType;

```

```

32 import javax.xml.bind.annotation.XmlElementRef;
33 import javax.xml.bind.annotation.XmlAttribute;
34 import javax.xml.bind.annotation.XmlSchema;
35 import javax.xml.bind.annotation.XmlAccessorType;
36 import javax.xml.bind.annotation.XmlSchemaTypes;
37 import java.lang.annotation.Target;
38 import java.lang.annotation.Retention;
39
40 import static java.lang.annotation.RetentionPolicy.RUNTIME;
41 import static java.lang.annotation.ElementType.FIELD;
42 import static java.lang.annotation.ElementType.METHOD;
43 import static java.lang.annotation.ElementType.TYPE;
44 import static java.lang.annotation.ElementType.PARAMETER;
45 import static java.lang.annotation.ElementType.PACKAGE;
46
47
48 /**
49  * Use an adapter that implements {@link XmlAdapter} for custom marshaling.
50  *
51  * <p> <b> Usage: </b> </p>
52  *
53  * <p> The <tt>@XmlJavaTypeAdapter</tt> annotation can be used with the
54  * following program elements:
55  * <ul>
56  *   <li> a JavaBean property </li>
57  *   <li> field </li>
58  *   <li> parameter </li>
59  *   <li> package </li>
60  *   <li> from within {@link XmlJavaTypeAdapters} </li>
61  * </ul>
62  *
63  * <p> When <tt>@XmlJavaTypeAdapter</tt> annotation is defined on a
64  * class, it applies to all references to the class.
65  * <p> When <tt>@XmlJavaTypeAdapter</tt> annotation is defined at the
66  * package level it applies to all references from within the package
67  * to <tt>@XmlJavaTypeAdapter.type()</tt>.
68  * <p> When <tt>@XmlJavaTypeAdapter</tt> annotation is defined on the
69  * field, property or parameter, then the annotation applies to the
70  * field, property or the parameter only.
71  * <p> A <tt>@XmlJavaTypeAdapter</tt> annotation on a field, property
72  * or parameter overrides the <tt>@XmlJavaTypeAdapter</tt> annotation
73  * associated with the class being referenced by the field, property
74  * or parameter.
75  * <p> A <tt>@XmlJavaTypeAdapter</tt> annotation on a class overrides
76  * the <tt>@XmlJavaTypeAdapter</tt> annotation specified at the
77  * package level for that class.
78  *
79  * <p> This annotation can be used with the following other annotations:
80  * {@link XmlElement}, {@link XmlAttribute}, {@link XmlElementRef},
81  * {@link XmlElementRefs}, {@link XmlAnyElement}. This can also be
82  * used at the package level with the following annotations:
83  * {@link XmlAccessorType}, {@link XmlSchema}, {@link XmlSchemaType},
84  * {@link XmlSchemaTypes}.

```

```

85  *
86  * <p><b> Example: </b> See example in {@link XmlAdapter}
87  *
88  * @author <ul><li>Sekhar Vajjhala, Sun Microsystems Inc.</li> <li> Kohsuke Kawaguchi, Sun
      Microsystems Inc.</li></ul>
89  * @since JAXB2.0
90  * @see XmlAdapter
91  */
92
93 @Retention(RUNTIME) @Target({PACKAGE, FIELD, METHOD, TYPE, PARAMETER})
94 public @interface XmlJavaTypeAdapter {
95     /**
96      * Points to the class that converts a value type to a bound type or vice versa.
97      * See {@link XmlAdapter} for more details.
98      */
99     Class<? extends XmlAdapter> value();
100
101     /**
102      * If this annotation is used at the package level, then value of
103      * the type() must be specified.
104      */
105
106     Class type() default DEFAULT.class;
107
108     /**
109      * Used in {@link XmlJavaTypeAdapter#type()} to
110      * signal that the type be inferred from the signature
111      * of the field, property, parameter or the class.
112      */
113
114     static final class DEFAULT {}
115
116 }

```

#### 4.6 XmlJavaTypeAdapters.java

Secuencia 75: javax/xml/bind/annotation/adapters/XmlJavaTypeAdapters.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *

```



```

17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.annotation.adapters;
27
28 import static java.lang.annotation.ElementType.PACKAGE;
29 import java.lang.annotation.Retention;
30 import static java.lang.annotation.RetentionPolicy.RUNTIME;
31 import java.lang.annotation.Target;
32
33 /**
34  * <p>
35  * A container for multiple {@link XmlJavaTypeAdapter} annotations.
36  *
37  * <p> Multiple annotations of the same type are not allowed on a program
38  * element. This annotation therefore serves as a container annotation
39  * for multiple &#64;XmlJavaTypeAdapter as follows:
40  *
41  * <pre>
42  * &#64;XmlJavaTypeAdapters ({ @XmlJavaTypeAdapter(...),@XmlJavaTypeAdapter(...) })
43  * </pre>
44  *
45  * <p>The <tt>@XmlJavaTypeAdapters</tt> annotation is useful for
46  * defining {@link XmlJavaTypeAdapter} annotations for different types
47  * at the package level.
48  *
49  * <p>See "Package Specification" in javax.xml.bind.package javadoc for
50  * additional common information.</p>
51  *
52  * @author <ul><li>Sekhar Vajjhala, Sun Microsystems, Inc.</li></ul>
53  * @see XmlJavaTypeAdapter
54  * @since JAXB2.0
55  */
56 @Retention(RUNTIME) @Target({PACKAGE})
57 public @interface XmlJavaTypeAdapters {
58     /**
59      * Collection of {@link XmlJavaTypeAdapter} annotations
60      */
61     XmlJavaTypeAdapter[] value();
62 }

```

## 5 Xml Bind Attachment (javax.xml.bind.attachment package)

### 5.1 AttachmentMarshaller.java

Secuencia 76: javax/xml/bind/attachment/AttachmentMarshaller.java

```

1  /*

```

```
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.attachment;
27
28 import javax.activation.DataHandler;
29 import javax.xml.bind.Marshaller;
30
31 /**
32  * <p>Enable JAXB marshalling to optimize storage of binary data.</p>
33  *
34  * <p>This API enables an efficient cooperative creation of optimized
35  * binary data formats between a JAXB marshalling process and a MIME-based package
36  * processor. A JAXB implementation marshals the root body of a MIME-based package,
37  * delegating the creation of referenceable MIME parts to
38  * the MIME-based package processor that implements this abstraction.</p>
39  *
40  * <p>XOP processing is enabled when {@link #isXOPPackage()} is true.
41  * See {@link #addMtomAttachment(DataHandler, String, String)} for details.
42  * </p>
43  *
44  * <p>WS-I Attachment Profile 1.0 is supported by
45  * {@link #addSwaRefAttachment(DataHandler)} being called by the
46  * marshaller for each JAXB property related to
47  * {http://ws-i.org/profiles/basic/1.1/xsd}swaRef.</p>
48  *
49  *
50  * @author Marc Hadley
51  * @author Kohsuke Kawaguchi
52  * @author Joseph Fialli
53  * @since JAXB 2.0
54  *
```

```

55 * @see Marshaller#setAttachmentMarshaller(AttachmentMarshaller)
56 *
57 * @see <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/">XML-binary Optimized Packaging</a>
58 * @see <a href="http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html">WS-I
    Attachments Profile Version 1.0.</a>
59 */
60 public abstract class AttachmentMarshaller {
61
62     /**
63      * <p>Consider MIME content <code>data</code> for optimized binary storage as an attachment.
64      *
65      * <p>
66      * This method is called by JAXB marshal process when {@link #isXOPPackage()} is
67      * <code>true</code>, for each element whose datatype is "base64Binary", as described in
68      * Step 3 in
69      * <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/#creating_xop_packages">Creating XOP
        Packages</a>.
70      *
71      * <p>
72      * The method implementor determines whether <code>data</code> shall be attached separately
73      * or inlined as base64Binary data. If the implementation chooses to optimize the storage
74      * of the binary data as a MIME part, it is responsible for attaching <code>data</code> to the
75      * MIME-based package, and then assigning an unique content-id, cid, that identifies
76      * the MIME part within the MIME message. This method returns the cid,
77      * which enables the JAXB marshaller to marshal a XOP element that refers to that cid in place
78      * of marshalling the binary data. When the method returns null, the JAXB marshaller
79      * inlines <code>data</code> as base64binary data.
80      *
81      * <p>
82      * The caller of this method is required to meet the following constraint.
83      * If the element info:et item containing <code>data</code> has the attribute
84      * <code>xmime:contentType</code> or if the JAXB property/field representing
85      * <code>data</code> is annotated with a known MIME type,
86      * <code>data.getContentType()</code> should be set to that MIME type.
87      *
88      * <p>
89      * The <code>elementNamespace</code> and <code>elementLocalName</code>
90      * parameters provide the
91      * context that contains the binary data. This information could
92      * be used by the MIME-based package processor to determine if the
93      * binary data should be inlined or optimized as an attachment.
94      *
95      * @param data
96      *     represents the data to be attached. Must be non-null.
97      * @param elementNamespace
98      *     the namespace URI of the element that encloses the base64Binary data.
99      *     Can be empty but never null.
100     * @param elementLocalName
101     *     The local name of the element. Always a non-null valid string.
102     *
103     * @return
104     *     a valid content-id URI (see <a href="http://www.w3.org/TR/xop10/#RFC2387">RFC 2387</a>)
        that identifies the attachment containing <code>data</code>.

```

```

105     * Otherwise, null if the attachment was not added and should instead be inlined in the
106     * message.
107     * @see <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/">XML-binary Optimized Packaging
108     * </a>
109     * @see <a href="http://www.w3.org/TR/xml-media-types/">Describing Media Content of Binary Data
110     * in XML</a>
111     */
112     public abstract String addMtomAttachment(DataHandler data, String elementNamespace, String
113         elementLocalName);
114
115     /**
116     * <p>Consider binary <code>data</code> for optimized binary storage as an attachment.
117     *
118     * <p>Since content type is not known, the attachment's MIME content type must be set to "
119     * application/octet-stream".</p>
120     *
121     * <p>
122     * The <code>elementNamespace</code> and <code>elementLocalName</code>
123     * parameters provide the
124     * context that contains the binary data. This information could
125     * be used by the MIME-based package processor to determine if the
126     * binary data should be inlined or optimized as an attachment.
127     *
128     * @param data
129     *     represents the data to be attached. Must be non-null. The actual data region is
130     *     specified by <tt>(data,offset,length)</tt> tuple.
131     *
132     * @param offset
133     *     The offset within the array of the first byte to be read;
134     *     must be non-negative and no larger than array.length
135     *
136     * @param length
137     *     The number of bytes to be read from the given array;
138     *     must be non-negative and no larger than array.length
139     *
140     * @param mimeType
141     *     If the data has an associated MIME type known to JAXB, that is passed
142     *     as this parameter. If none is known, "application/octet-stream".
143     *     This parameter may never be null.
144     *
145     * @param elementNamespace
146     *     the namespace URI of the element that encloses the base64Binary data.
147     *     Can be empty but never null.
148     *
149     * @param elementLocalName
150     *     The local name of the element. Always a non-null valid string.
151     *
152     * @return content-id URI, cid, to the attachment containing
153     *     <code>data</code> or null if data should be inlined.
154     *
155     * @see #addMtomAttachment(DataHandler, String, String)
156     */

```

```

153 public abstract String addMtomAttachment(byte[] data, int offset, int length, String mimeType,
    String elementNamespace, String elementLocalName);
154
155 /**
156  * <p>Read-only property that returns true if JAXB marshaller should enable XOP creation.</p>
157  *
158  * <p>This value must not change during the marshalling process. When this
159  * value is true, the <code>addMtomAttachment(...)</code> method
160  * is invoked when the appropriate binary datatypes are encountered by
161  * the marshal process.</p>
162  *
163  * <p>Marshaller.marshal() must throw IllegalStateException if this value is <code>true</code>
164  * and the XML content to be marshalled violates Step 1 in
165  * <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/#creating_xop_packages">Creating XOP
    Pacakges</a>
166  * http://www.w3.org/TR/2005/REC-xop10-20050125/#creating_xop_packages.
167  * <i>"Ensure the Original XML Infoset contains no element information item with a
168  * [namespace name] of "http://www.w3.org/2004/08/xop/include" and a [local name] of Include"</i>
    i>
169  *
170  * <p>When this method returns true and during the marshal process
171  * at least one call to <code>addMtomAttachment(...)</code> returns
172  * a content-id, the MIME-based package processor must label the
173  * root part with the application/xop+xml media type as described in
174  * Step 5 of
175  * <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/#creating_xop_packages">Creating XOP
    Pacakges</a>.<p>
176  *
177  * @return true when MIME context is a XOP Package.
178  */
179 public boolean isXOPPackage() { return false; }
180
181 /**
182  * <p>Add MIME <code>data</code> as an attachment and return attachment's content-id, cid.</p>
183  *
184  * <p>
185  * This method is called by JAXB marshal process for each element/attribute typed as
186  * {http://ws-i.org/profiles/basic/1.1/xsd}swaRef. The MIME-based package processor
187  * implementing this method is responsible for attaching the specified data to a
188  * MIME attachment, and generating a content-id, cid, that uniquely identifies the attachment
189  * within the MIME-based package.
190  *
191  * <p>Caller inserts the returned content-id, cid, into the XML content being marshalled.</p>
192  *
193  * @param data
194  *     represents the data to be attached. Must be non-null.
195  * @return
196  *     must be a valid URI used as cid. Must satisfy Conformance Requirement R2928 from
197  *     <a href="http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html#
    Referencing_Attachments_from_the_SOAP_Envelope">WS-I Attachments Profile Version 1.0.</a>
198  */
199 public abstract String addSwaRefAttachment(DataHandler data);
200 }

```

## 5.2 AttachmentUnmarshaller.java

Secuencia 77: javax/xml/bind/attachment/AttachmentUnmarshaller.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.attachment;
27
28 import javax.activation.DataHandler;
29
30 /**
31  * <p>Enables JAXB unmarshalling of a root document containing optimized binary data formats.</p>
32  *
33  * <p>This API enables an efficient cooperative processing of optimized
34  * binary data formats between a JAXB 2.0 implementation and MIME-based package
35  * processor (MTOM/XOP and WS-I AP 1.0). JAXB unmarshals the body of a package, delegating the
36  * understanding of the packaging format being used to a MIME-based
37  * package processor that implements this abstract class.</p>
38  *
39  * <p>This abstract class identifies if a package requires XOP processing, {@link #isXOPPackage()}
40  * and provides retrieval of binary content stored as attachments by content-id.</p>
41  *
42  * <h2>Identifying the content-id, cid, to pass to <code>getAttachment*(String cid)</code></h2>
43  * <ul>
44  * <li>
45  * For XOP processing, the info:et representation of the cid is described
46  * in step 2a in
47  * <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/#interpreting_xop_packages">Section 3.2
48  * Interpreting XOP Packages</a>
49  * </li>
50  * <li>

```

```

49  * For WS-I AP 1.0, the cid is identified as an element or attribute of
50  * type <code>ref:swaRef </code> specified in
51  * <a href="http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html#
    Referencing_Attachments_from_the_SOAP_Envelope">Section 4.4 Referencing Attachments from the
    SOAP Envelope</a>
52  * </li>
53  * </ul>
54  *
55  * @author Marc Hadley
56  * @author Kohsuke Kawaguchi
57  * @author Joseph Fialli
58  *
59  * @since JAXB 2.0
60  *
61  * @see javax.xml.bind.Unmarshaler#setAttachmentUnmarshaller(AttachmentUnmarshaller)
62  *
63  * @see <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/">XML-binary Optimized Packaging</a>
64  * @see <a href="http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html">WS-I
    Attachments Profile Version 1.0.</a>
65  * @see <a href="http://www.w3.org/TR/xml-media-types/">Describing Media Content of Binary Data in
    XML</a>
66  */
67  public abstract class AttachmentUnmarshaller {
68      /**
69       * <p>Lookup MIME content by content-id, <code>cid</code>, and return as a {@link DataHandler
        }.</p>
70       *
71       * <p>The returned <code>DataHandler</code> instance must be configured
72       * to meet the following required mapping constraint.
73       * <table border="2" rules="all" cellpadding="4">
74       *   <thead>
75       *     <tr>
76       *       <th align="center" colspan="2">
77       *         Required Mappings between MIME and Java Types
78       *       </tr>
79       *     <tr>
80       *       <th>MIME Type</th>
81       *       <th>Java Type</th>
82       *     </tr>
83       *     <tr>
84       *       <th><code>DataHandler.getContentType()</code></th>
85       *       <th><code>instanceof DataHandler.getContent()</code></th>
86       *     </tr>
87       *   </thead>
88       *   <tbody>
89       *     <tr>
90       *       <td>image/gif</td>
91       *       <td>java.awt.Image</td>
92       *     </tr>
93       *     <tr>
94       *       <td>image/jpeg</td>
95       *       <td>java.awt.Image</td>
96       *     </tr>

```

```

97      *      <tr>
98      *          <td>text/xml or application/xml</td>
99      *          <td>javax.xml.transform.Source</td>
100     *      </tr>
101     *  </tbody>
102     *  </table>
103     *  Note that it is allowable to support additional mappings.</p>
104     *
105     *  @param cid It is expected to be a valid lexical form of the XML Schema
106     *  <code>xs:anyURI</code> datatype. If <code>{@link #isXOPPackage()}</code>
107     *  ==true</code>, it must be a valid URI per the <code>cid:</code> URI scheme (see <a href="http
108     *  ://www.ietf.org/rfc/rfc2387.txt">RFC 2387</a>)
109     *
110     *  @return
111     *      a {@link DataHandler} that represents the MIME attachment.
112     *
113     *  @throws IllegalArgumentException if the attachment for the given cid is not found.
114     */
115     public abstract DataHandler getAttachmentAsDataHandler(String cid);
116
117     /**
118     *  <p>Retrieve the attachment identified by content-id, <code>cid</code>, as a <tt>byte[]</tt>
119     *  </p>.
120     *
121     *  @param cid It is expected to be a valid lexical form of the XML Schema
122     *  <code>xs:anyURI</code> datatype. If <code>{@link #isXOPPackage()}</code>
123     *  ==true</code>, it must be a valid URI per the <code>cid:</code> URI scheme (see <a href="
124     *  http://www.ietf.org/rfc/rfc2387.txt">RFC 2387</a>)
125     *
126     *  @return byte[] representation of attachment identified by cid.
127     *
128     *  @throws IllegalArgumentException if the attachment for the given cid is not found.
129     */
130     public abstract byte[] getAttachmentAsByteArray(String cid);
131
132     /**
133     *  <p>Read-only property that returns true if JAXB unmarshaller needs to perform XOP processing
134     *  .</p>
135     *
136     *  <p>This method returns <code>true</code> when the constraints specified
137     *  in <a href="http://www.w3.org/TR/2005/REC-xop10-20050125/#identifying_xop_documents">
138     *  Identifying XOP Documents</a> are met.
139     *  This value must not change during the unmarshalling process.</p>
140     *
141     *  @return true when MIME context is a XOP Document.
142     */
143     public boolean isXOPPackage() { return false; }
144 }

```

## 6 Xml Bind Helpers (javax.xml.bind.helpers package)

### 6.1 AbstractMarshallerImpl.java



Secuencia 78: javax/xml/bind/helpers/AbstractMarshallerImpl.java

```
1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.helpers;
27
28 import javax.xml.bind.JAXBException;
29 import javax.xml.bind.Marshaller;
30 import javax.xml.bind.PropertyException;
31 import javax.xml.bind.ValidationEventHandler;
32 import javax.xml.bind.annotation.adapters.XmlAdapter;
33 import javax.xml.bind.attachment.AttachmentMarshaller;
34 import javax.xml.stream.XMLEventWriter;
35 import javax.xml.stream.XMLStreamWriter;
36 import javax.xml.transform.dom.DOMResult;
37 import javax.xml.transform.sax.SAXResult;
38 import javax.xml.transform.stream.StreamResult;
39 import javax.xml.validation.Schema;
40 import java.io.UnsupportedEncodingException;
41 import java.io.File;
42 import java.io.OutputStream;
43 import java.io.FileOutputStream;
44 import java.io.BufferedOutputStream;
45 import java.io.IOException;
46 // J2SE1.4 feature
47 // import java.nio.charset.Charset;
48 // import java.nio.charset.UnsupportedCharsetException;
49
50 /**
51 * Partial default <tt>Marshaller</tt> implementation.
52 *
```

```

53  * <p>
54  * This class provides a partial default implementation for the
55  * {@link javax.xml.bind.Marshaller} interface.
56  *
57  * <p>
58  * The only methods that a JAXB Provider has to implement are
59  * {@link Marshaller#marshal(Object, javax.xml.transform.Result) marshal(Object, javax.xml.
    transform.Result)},
60  * {@link Marshaller#marshal(Object, javax.xml.transform.Result) marshal(Object, javax.xml.stream.
    XMLStreamWriter)}, and
61  * {@link Marshaller#marshal(Object, javax.xml.transform.Result) marshal(Object, javax.xml.stream.
    XMLEventWriter)}.
62  *
63  * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li></ul>
64  * @see javax.xml.bind.Marshaller
65  * @since JAXB1.0
66  */
67  public abstract class AbstractMarshallerImpl implements Marshaller
68  {
69      /** handler that will be used to process errors and warnings during marshal */
70      private ValidationEventHandler eventHandler =
71          new DefaultValidationEventHandler();
72
73      //J2SE1.4 feature
74      //private Charset encoding = null;
75
76      /** store the value of the encoding property. */
77      private String encoding = "UTF-8";
78
79      /** store the value of the schemaLocation property. */
80      private String schemaLocation = null;
81
82      /** store the value of the noNamespaceSchemaLocation property. */
83      private String noNSSchemaLocation = null;
84
85      /** store the value of the formattedOutput property. */
86      private boolean formattedOutput = false;
87
88      /** store the value of the fragment property. */
89      private boolean fragment = false;
90
91      public final void marshal( Object obj, java.io.OutputStream os )
92          throws JAXBException {
93
94          checkNotNull( obj, "obj", os, "os" );
95          marshal( obj, new StreamResult(os) );
96      }
97
98      public void marshal(Object jaxbElement, File output) throws JAXBException {
99          checkNotNull(jaxbElement, "jaxbElement", output, "output" );
100         try {
101             OutputStream os = new BufferedOutputStream(new FileOutputStream(output));
102             try {

```

```
103         marshal( jaxbElement, new StreamResult(os) );
104     } finally {
105         os.close();
106     }
107 } catch (IOException e) {
108     throw new JAXBException(e);
109 }
110 }
111
112 public final void marshal( Object obj, java.io.Writer w )
113     throws JAXBException {
114
115     checkNotNull( obj, "obj", w, "writer" );
116     marshal( obj, new StreamResult(w) );
117 }
118
119 public final void marshal( Object obj, org.xml.sax.ContentHandler handler )
120     throws JAXBException {
121
122     checkNotNull( obj, "obj", handler, "handler" );
123     marshal( obj, new SAXResult(handler) );
124 }
125
126 public final void marshal( Object obj, org.w3c.dom.Node node )
127     throws JAXBException {
128
129     checkNotNull( obj, "obj", node, "node" );
130     marshal( obj, new DOMResult(node) );
131 }
132
133 /**
134  * By default, the getNode method is unsupported and throw
135  * an {@link java.lang.UnsupportedOperationException}.
136  *
137  * Implementations that choose to support this method must
138  * override this method.
139  */
140 public org.w3c.dom.Node getNode( Object obj ) throws JAXBException {
141
142     checkNotNull( obj, "obj", Boolean.TRUE, "foo" );
143
144     throw new UnsupportedOperationException();
145 }
146
147 /**
148  * Convenience method for getting the current output encoding.
149  *
150  * @return the current encoding or "UTF-8" if it hasn't been set.
151  */
152 protected String getEncoding() {
153     return encoding;
154 }
155
```

```
156 /**
157  * Convenience method for setting the output encoding.
158  *
159  * @param encoding a valid encoding as specified in the Marshaller class
160  * documentation
161  */
162 protected void setEncoding( String encoding ) {
163     this.encoding = encoding;
164 }
165
166 /**
167  * Convenience method for getting the current schemaLocation.
168  *
169  * @return the current schemaLocation or null if it hasn't been set
170  */
171 protected String getSchemaLocation() {
172     return schemaLocation;
173 }
174
175 /**
176  * Convenience method for setting the schemaLocation.
177  *
178  * @param location the schemaLocation value
179  */
180 protected void setSchemaLocation( String location ) {
181     schemaLocation = location;
182 }
183
184 /**
185  * Convenience method for getting the current noNamespaceSchemaLocation.
186  *
187  * @return the current noNamespaceSchemaLocation or null if it hasn't
188  * been set
189  */
190 protected String getNoNSSchemaLocation() {
191     return noNSSchemaLocation;
192 }
193
194 /**
195  * Convenience method for setting the noNamespaceSchemaLocation.
196  *
197  * @param location the noNamespaceSchemaLocation value
198  */
199 protected void setNoNSSchemaLocation( String location ) {
200     noNSSchemaLocation = location;
201 }
202
203 /**
204  * Convenience method for getting the formatted output flag.
205  *
206  * @return the current value of the formatted output flag or false if
207  * it hasn't been set.
208  */
```

```
209     protected boolean isFormattedOutput() {
210         return formattedOutput;
211     }
212
213     /**
214      * Convenience method for setting the formatted output flag.
215      *
216      * @param v value of the formatted output flag.
217      */
218     protected void setFormattedOutput( boolean v ) {
219         formattedOutput = v;
220     }
221
222
223     /**
224      * Convenience method for getting the fragment flag.
225      *
226      * @return the current value of the fragment flag or false if
227      * it hasn't been set.
228      */
229     protected boolean isFragment() {
230         return fragment;
231     }
232
233     /**
234      * Convenience method for setting the fragment flag.
235      *
236      * @param v value of the fragment flag.
237      */
238     protected void setFragment( boolean v ) {
239         fragment = v;
240     }
241
242
243     static String[] aliases = {
244         "UTF-8", "UTF8",
245         "UTF-16", "Unicode",
246         "UTF-16BE", "UnicodeBigUnmarked",
247         "UTF-16LE", "UnicodeLittleUnmarked",
248         "US-ASCII", "ASCII",
249         "TIS-620", "TIS620",
250
251         // taken from the project-X parser
252         "ISO-10646-UCS-2", "Unicode",
253
254         "EBCDIC-CP-US", "cp037",
255         "EBCDIC-CP-CA", "cp037",
256         "EBCDIC-CP-NL", "cp037",
257         "EBCDIC-CP-WT", "cp037",
258
259         "EBCDIC-CP-DK", "cp277",
260         "EBCDIC-CP-NO", "cp277",
261         "EBCDIC-CP-FI", "cp278",
```

```

262     "EBCDIC-CP-SE", "cp278",
263
264     "EBCDIC-CP-IT", "cp280",
265     "EBCDIC-CP-ES", "cp284",
266     "EBCDIC-CP-GB", "cp285",
267     "EBCDIC-CP-FR", "cp297",
268
269     "EBCDIC-CP-AR1", "cp420",
270     "EBCDIC-CP-HE", "cp424",
271     "EBCDIC-CP-BE", "cp500",
272     "EBCDIC-CP-CH", "cp500",
273
274     "EBCDIC-CP-ROECE", "cp870",
275     "EBCDIC-CP-YU", "cp870",
276     "EBCDIC-CP-IS", "cp871",
277     "EBCDIC-CP-AR2", "cp918",
278
279     // IANA also defines two that JDK 1.2 doesn't handle:
280     // EBCDIC-CP-GR      --> CP423
281     // EBCDIC-CP-TR      --> CP905
282 };
283
284 /**
285  * Gets the corresponding Java encoding name from an IANA name.
286  *
287  * This method is a helper method for the derived class to convert
288  * encoding names.
289  *
290  * @exception UnsupportedOperationException
291  *     If this implementation couldn't find the Java encoding name.
292  */
293 protected String getJavaEncoding( String encoding ) throws UnsupportedOperationException {
294     try {
295         "1".getBytes(encoding);
296         return encoding;
297     } catch( UnsupportedOperationException e ) {
298         // try known alias
299         for( int i=0; i<aliases.length; i+=2 ) {
300             if(encoding.equals(aliases[i])) {
301                 "1".getBytes(aliases[i+1]);
302                 return aliases[i+1];
303             }
304         }
305
306         throw new UnsupportedOperationException(encoding);
307     }
308     /* J2SE1.4 feature
309     try {
310         this.encoding = Charset.forName( _encoding );
311     } catch( UnsupportedCharsetException uce ) {
312         throw new JAXBException( uce );
313     }
314     */

```

```
315     }
316
317     /**
318     * Default implementation of the setProperty method handles
319     * the four defined properties in Marshaller. If a provider
320     * needs to handle additional properties, it should override
321     * this method in a derived class.
322     */
323     public void setProperty( String name, Object value )
324         throws PropertyException {
325
326         if( name == null ) {
327             throw new IllegalArgumentException(
328                 Messages.format( Messages.MUST_NOT_BE_NULL, "name" ) );
329         }
330
331         // recognize and handle four pre-defined properties.
332         if( JAXB_ENCODING.equals(name) ) {
333             checkString( name, value );
334             setEncoding( (String)value );
335             return;
336         }
337         if( JAXB_FORMATTED_OUTPUT.equals(name) ) {
338             checkBoolean( name, value );
339             setFormattedOutput((Boolean) value );
340             return;
341         }
342         if( JAXB_NO_NAMESPACE_SCHEMA_LOCATION.equals(name) ) {
343             checkString( name, value );
344             setNoNSSchemaLocation( (String)value );
345             return;
346         }
347         if( JAXB_SCHEMA_LOCATION.equals(name) ) {
348             checkString( name, value );
349             setSchemaLocation( (String)value );
350             return;
351         }
352         if( JAXB_FRAGMENT.equals(name) ) {
353             checkBoolean(name, value);
354             setFragment((Boolean) value );
355             return;
356         }
357
358         throw new PropertyException(name, value);
359     }
360
361     /**
362     * Default implementation of the getProperty method handles
363     * the four defined properties in Marshaller. If a provider
364     * needs to support additional provider specific properties,
365     * it should override this method in a derived class.
366     */
367     public Object getProperty( String name )
```

```

368     throws PropertyException {
369
370     if( name == null ) {
371         throw new IllegalArgumentException(
372             Messages.format( Messages.MUST_NOT_BE_NULL, "name" ) );
373     }
374
375     // recognize and handle four pre-defined properties.
376     if( JAXB_ENCODING.equals(name) )
377         return getEncoding();
378     if( JAXB_FORMATTED_OUTPUT.equals(name) )
379         return isFormattedOutput()?Boolean.TRUE:Boolean.FALSE;
380     if( JAXB_NO_NAMESPACE_SCHEMA_LOCATION.equals(name) )
381         return getNonSSchemaLocation();
382     if( JAXB_SCHEMA_LOCATION.equals(name) )
383         return getSchemaLocation();
384     if( JAXB_FRAGMENT.equals(name) )
385         return isFragment()?Boolean.TRUE:Boolean.FALSE;
386
387     throw new PropertyException(name);
388 }
389 /**
390  * @see javax.xml.bind.Marshaller#getEventHandler()
391  */
392 public ValidationEventHandler getEventHandler() throws JAXBException {
393     return eventHandler;
394 }
395
396 /**
397  * @see javax.xml.bind.Marshaller#setEventHandler(ValidationEventHandler)
398  */
399 public void setEventHandler(ValidationEventHandler handler)
400     throws JAXBException {
401
402     if( handler == null ) {
403         eventHandler = new DefaultValidationEventHandler();
404     } else {
405         eventHandler = handler;
406     }
407 }
408
409
410
411
412 /**
413  * assert that the given object is a Boolean
414  */
415 private void checkBoolean( String name, Object value ) throws PropertyException {
416     if(!(value instanceof Boolean))
417         throw new PropertyException(
418             Messages.format( Messages.MUST_BE_BOOLEAN, name ) );
419 }
420

```



```
421  /*
422   * assert that the given object is a String
423   */
424  private void checkString( String name, Object value ) throws PropertyException {
425      if(!(value instanceof String))
426          throw new PropertyException(
427              Messages.format( Messages.MUST_BE_STRING, name ) );
428  }
429
430  /*
431   * assert that the parameters are not null
432   */
433  private void checkNotNull( Object o1, String o1Name,
434                          Object o2, String o2Name ) {
435
436      if( o1 == null ) {
437          throw new IllegalArgumentException(
438              Messages.format( Messages.MUST_NOT_BE_NULL, o1Name ) );
439      }
440      if( o2 == null ) {
441          throw new IllegalArgumentException(
442              Messages.format( Messages.MUST_NOT_BE_NULL, o2Name ) );
443      }
444  }
445
446  public void marshal(Object obj, XMLEventWriter writer)
447      throws JAXBException {
448
449      throw new UnsupportedOperationException();
450  }
451
452  public void marshal(Object obj, XMLStreamWriter writer)
453      throws JAXBException {
454
455      throw new UnsupportedOperationException();
456  }
457
458  public void setSchema(Schema schema) {
459      throw new UnsupportedOperationException();
460  }
461
462  public Schema getSchema() {
463      throw new UnsupportedOperationException();
464  }
465
466  public void setAdapter(XmlAdapter adapter) {
467      if(adapter==null)
468          throw new IllegalArgumentException();
469      setAdapter((Class)adapter.getClass(),adapter);
470  }
471
472  public <A extends XmlAdapter> void setAdapter(Class<A> type, A adapter) {
473      throw new UnsupportedOperationException();
```

```

474     }
475
476     public <A extends XmlAdapter> A getAdapter(Class<A> type) {
477         throw new UnsupportedOperationException();
478     }
479
480     public void setAttachmentMarshaller(AttachmentMarshaller am) {
481         throw new UnsupportedOperationException();
482     }
483
484     public AttachmentMarshaller getAttachmentMarshaller() {
485         throw new UnsupportedOperationException();
486     }
487
488     public void setListener(Listener listener) {
489         throw new UnsupportedOperationException();
490     }
491
492     public Listener getListener() {
493         throw new UnsupportedOperationException();
494     }
495 }

```

## 6.2 AbstractUnmarshallerImpl.java

Secuencia 79: javax/xml/bind/helpers/AbstractUnmarshallerImpl.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.helpers;
27

```

```
28 import org.xml.sax.InputSource;
29 import org.xml.sax.SAXException;
30 import org.xml.sax.XMLReader;
31 import org.w3c.dom.Node;
32
33 import javax.xml.bind.JAXBException;
34 import javax.xml.bind.PropertyException;
35 import javax.xml.bind.UnmarshalException;
36 import javax.xml.bind.Unmarshaler;
37 import javax.xml.bind.ValidationEventHandler;
38 import javax.xml.bind.JAXBElement;
39 import javax.xml.bind.annotation.adapters.XmlAdapter;
40 import javax.xml.bind.attachment.AttachmentUnmarshaller;
41 import javax.xml.parsers.ParserConfigurationException;
42 import javax.xml.parsers.SAXParserFactory;
43 import javax.xml.stream.XMLEventReader;
44 import javax.xml.stream.XMLStreamReader;
45 import javax.xml.transform.Source;
46 import javax.xml.transform.dom.DOMSource;
47 import javax.xml.transform.sax.SAXSource;
48 import javax.xml.transform.stream.StreamSource;
49 import javax.xml.validation.Schema;
50 import java.io.File;
51 import java.io.Reader;
52 import java.net.MalformedURLException;
53 import java.net.URL;
54
55 /**
56  * Partial default <tt>Unmarshaller</tt> implementation.
57  *
58  * <p>
59  * This class provides a partial default implementation for the
60  * {@link javax.xml.bind.Unmarshaler} interface.
61  *
62  * <p>
63  * A JAXB Provider has to implement five methods (getUnmarshallerHandler,
64  * unmarshal(Node), unmarshal(XMLReader,InputSource),
65  * unmarshal(XMLStreamReader), and unmarshal(XMLEventReader).
66  *
67  * @author <ul>
68  * <li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li>
69  * </ul>
70  * @see javax.xml.bind.Unmarshaler
71  * @since JAXB1.0
72  */
73 public abstract class AbstractUnmarshallerImpl implements Unmarshaller
74 {
75     /** handler that will be used to process errors and warnings during unmarshal */
76     private ValidationEventHandler eventHandler =
77         new DefaultValidationEventHandler();
78
79     /** whether or not the unmarshaller will validate */
80     protected boolean validating = false;
```

```

81
82  /**
83   * XMLReader that will be used to parse a document.
84   */
85  private XMLReader reader = null;
86
87  /**
88   * Obtains a configured XMLReader.
89   *
90   * This method is used when the client-specified
91   * {@link SAXSource} object doesn't have XMLReader.
92   *
93   * {@link Unmarshaller} is not re-entrant, so we will
94   * only use one instance of XMLReader.
95   */
96  protected XMLReader getXMLReader() throws JAXBException {
97      if(reader==null) {
98          try {
99              SAXParserFactory parserFactory;
100              parserFactory = SAXParserFactory.newInstance();
101              parserFactory.setNamespaceAware(true);
102              // there is no point in asking a validation because
103              // there is no guarantee that the document will come with
104              // a proper schemaLocation.
105              parserFactory.setValidating(false);
106              reader = parserFactory.newSAXParser().getXMLReader();
107          } catch( ParserConfigurationException e ) {
108              throw new JAXBException(e);
109          } catch( SAXException e ) {
110              throw new JAXBException(e);
111          }
112      }
113      return reader;
114  }
115
116  public Object unmarshal( Source source ) throws JAXBException {
117      if( source == null ) {
118          throw new IllegalArgumentException(
119              Messages.format( Messages.MUST_NOT_BE_NULL, "source" ) );
120      }
121
122      if(source instanceof SAXSource)
123          return unmarshal( (SAXSource)source );
124      if(source instanceof StreamSource)
125          return unmarshal( streamSourceToInputSource((StreamSource)source));
126      if(source instanceof DOMSource)
127          return unmarshal( ((DOMSource)source).getNode() );
128
129      // we don't handle other types of Source
130      throw new IllegalArgumentException();
131  }
132
133  // use the client specified XMLReader contained in the SAXSource.

```

```
134 private Object unmarshal( SAXSource source ) throws JAXBException {
135
136     XMLReader r = source.getXMLReader();
137     if( r == null )
138         r = getXMLReader();
139
140     return unmarshal( r, source.getInputSource() );
141 }
142
143 /**
144  * Unmarshals an object by using the specified XMLReader and the InputSource.
145  *
146  * The callee should call the setErrorHandler method of the XMLReader
147  * so that errors are passed to the client-specified ValidationEventHandler.
148  */
149 protected abstract Object unmarshal( XMLReader reader, InputSource source ) throws
    JAXBException;
150
151 public final Object unmarshal( InputSource source ) throws JAXBException {
152     if( source == null ) {
153         throw new IllegalArgumentException(
154             Messages.format( Messages.MUST_NOT_BE_NULL, "source" ) );
155     }
156
157     return unmarshal( getXMLReader(), source );
158 }
159
160
161 private Object unmarshal( String url ) throws JAXBException {
162     return unmarshal( new InputSource(url) );
163 }
164
165 public final Object unmarshal( URL url ) throws JAXBException {
166     if( url == null ) {
167         throw new IllegalArgumentException(
168             Messages.format( Messages.MUST_NOT_BE_NULL, "url" ) );
169     }
170
171     return unmarshal( url.toExternalForm() );
172 }
173
174 public final Object unmarshal( File f ) throws JAXBException {
175     if( f == null ) {
176         throw new IllegalArgumentException(
177             Messages.format( Messages.MUST_NOT_BE_NULL, "file" ) );
178     }
179
180     try {
181         // copied from JAXP
182         String path = f.getAbsolutePath();
183         if (File.separatorChar != '/')
184             path = path.replace(File.separatorChar, '/');
185         if (!path.startsWith("/"))
```

```

186         path = "/" + path;
187         if (!path.endsWith("/") && f.isDirectory())
188             path = path + "/";
189         return unmarshal(new URL("file", "", path));
190     } catch( MalformedURLException e ) {
191         throw new IllegalArgumentException(e.getMessage());
192     }
193 }
194
195 public final Object unmarshal( java.io.InputStream is )
196     throws JAXBException {
197
198     if( is == null ) {
199         throw new IllegalArgumentException(
200             Messages.format( Messages.MUST_NOT_BE_NULL, "is" ) );
201     }
202
203     InputSource isrc = new InputSource( is );
204     return unmarshal( isrc );
205 }
206
207 public final Object unmarshal( Reader reader ) throws JAXBException {
208     if( reader == null ) {
209         throw new IllegalArgumentException(
210             Messages.format( Messages.MUST_NOT_BE_NULL, "reader" ) );
211     }
212
213     InputSource isrc = new InputSource( reader );
214     return unmarshal( isrc );
215 }
216
217
218 private static InputSource streamSourceToInputSource( StreamSource ss ) {
219     InputSource is = new InputSource();
220     is.setSystemId( ss.getSystemId() );
221     is.setByteStream( ss.getInputStream() );
222     is.setCharacterStream( ss.getReader() );
223
224     return is;
225 }
226
227
228 /**
229  * Indicates whether or not the Unmarshaller is configured to validate
230  * during unmarshal operations.
231  * <p>
232  * <i><b>Note:</b> I named this method isValidating() to stay in-line
233  * with JAXP, as opposed to naming it getValidating(). </i>
234  *
235  * @return true if the Unmarshaller is configured to validate during
236  *         unmarshal operations, false otherwise
237  * @throws JAXBException if an error occurs while retrieving the validating
238  *         flag

```

```
239     */
240     public boolean isValidating() throws JAXBException {
241         return validating;
242     }
243
244     /**
245      * Allow an application to register a validation event handler.
246      * <p>
247      * The validation event handler will be called by the JAXB Provider if any
248      * validation errors are encountered during calls to any of the
249      * <tt>unmarshal</tt> methods. If the client application does not register
250      * a validation event handler before invoking the unmarshal methods, then
251      * all validation events will be silently ignored and may result in
252      * unexpected behaviour.
253      *
254      * @param handler the validation event handler
255      * @throws JAXBException if an error was encountered while setting the
256      *         event handler
257      */
258     public void setEventHandler(ValidationEventHandler handler)
259         throws JAXBException {
260
261         if( handler == null ) {
262             errorHandler = new DefaultValidationEventHandler();
263         } else {
264             errorHandler = handler;
265         }
266     }
267
268     /**
269      * Specifies whether or not the Unmarshaller should validate during
270      * unmarshal operations. By default, the <tt>Unmarshaller</tt> does
271      * not validate.
272      * <p>
273      * This method may only be invoked before or after calling one of the
274      * unmarshal methods.
275      *
276      * @param validating true if the Unmarshaller should validate during
277      *         unmarshal, false otherwise
278      * @throws JAXBException if an error occurred while enabling or disabling
279      *         validation at unmarshal time
280      */
281     public void setValidating(boolean validating) throws JAXBException {
282         this.validating = validating;
283     }
284
285     /**
286      * Return the current event handler or the default event handler if one
287      * hasn't been set.
288      *
289      * @return the current ValidationEventHandler or the default event handler
290      *         if it hasn't been set
291      * @throws JAXBException if an error was encountered while getting the
```

```

292     *         current event handler
293     */
294     public ValidationEventHandler getEventHandler() throws JAXBException {
295         return eventHandler;
296     }
297
298
299     /**
300     * Creates an UnmarshalException from a SAXException.
301     *
302     * This is an utility method provided for the derived classes.
303     *
304     * <p>
305     * When a provider-implemented ContentHandler wants to throw a
306     * JAXBException, it needs to wrap the exception by a SAXException.
307     * If the unmarshaller implementation blindly wrap SAXException
308     * by JAXBException, such an exception will be a JAXBException
309     * wrapped by a SAXException wrapped by another JAXBException.
310     * This is silly.
311     *
312     * <p>
313     * This method checks the nested exception of SAXException
314     * and reduce those excessive wrapping.
315     *
316     * @return the resulting UnmarshalException
317     */
318     protected UnmarshalException createUnmarshalException( SAXException e ) {
319         // check the nested exception to see if it's an UnmarshalException
320         Exception nested = e.getException();
321         if(nested instanceof UnmarshalException)
322             return (UnmarshalException)nested;
323
324         if(nested instanceof RuntimeException)
325             // typically this is an unexpected exception,
326             // just throw it rather than wrap it, so that the full stack
327             // trace can be displayed.
328             throw (RuntimeException)nested;
329
330
331         // otherwise simply wrap it
332         if(nested!=null)
333             return new UnmarshalException(nested);
334         else
335             return new UnmarshalException(e);
336     }
337
338     /**
339     * Default implementation of the setProperty method always
340     * throws PropertyException since there are no required
341     * properties. If a provider needs to handle additional
342     * properties, it should override this method in a derived class.
343     */
344     public void setProperty( String name, Object value )

```



```
345     throws PropertyException {
346
347     if( name == null ) {
348         throw new IllegalArgumentException(
349             Messages.format( Messages.MUST_NOT_BE_NULL, "name" ) );
350     }
351
352     throw new PropertyException(name, value);
353 }
354
355 /**
356  * Default implementation of the getProperty method always
357  * throws PropertyException since there are no required
358  * properties. If a provider needs to handle additional
359  * properties, it should override this method in a derived class.
360  */
361 public Object getProperty( String name )
362     throws PropertyException {
363
364     if( name == null ) {
365         throw new IllegalArgumentException(
366             Messages.format( Messages.MUST_NOT_BE_NULL, "name" ) );
367     }
368
369     throw new PropertyException(name);
370 }
371
372 public Object unmarshal(XMLStreamReader reader) throws JAXBException {
373
374     throw new UnsupportedOperationException();
375 }
376
377 public Object unmarshal(XMLStreamReader reader) throws JAXBException {
378
379     throw new UnsupportedOperationException();
380 }
381
382 public <T> JAXBElement<T> unmarshal(Node node, Class<T> expectedType) throws JAXBException {
383     throw new UnsupportedOperationException();
384 }
385
386 public <T> JAXBElement<T> unmarshal(Source source, Class<T> expectedType) throws JAXBException
387     {
388     throw new UnsupportedOperationException();
389 }
390
391 public <T> JAXBElement<T> unmarshal(XMLStreamReader reader, Class<T> expectedType) throws
392     JAXBException {
393     throw new UnsupportedOperationException();
394 }
395
396 public <T> JAXBElement<T> unmarshal(XMLStreamReader reader, Class<T> expectedType) throws
397     JAXBException {
```

```

395         throw new UnsupportedOperationException();
396     }
397
398     public void setSchema(Schema schema) {
399         throw new UnsupportedOperationException();
400     }
401
402     public Schema getSchema() {
403         throw new UnsupportedOperationException();
404     }
405
406     public void setAdapter(XmlAdapter adapter) {
407         if(adapter==null)
408             throw new IllegalArgumentException();
409         setAdapter((Class)adapter.getClass(),adapter);
410     }
411
412     public <A extends XmlAdapter> void setAdapter(Class<A> type, A adapter) {
413         throw new UnsupportedOperationException();
414     }
415
416     public <A extends XmlAdapter> A getAdapter(Class<A> type) {
417         throw new UnsupportedOperationException();
418     }
419
420     public void setAttachmentUnmarshaller(AttachmentUnmarshaller au) {
421         throw new UnsupportedOperationException();
422     }
423
424     public AttachmentUnmarshaller getAttachmentUnmarshaller() {
425         throw new UnsupportedOperationException();
426     }
427
428     public void setListener(Listener listener) {
429         throw new UnsupportedOperationException();
430     }
431
432     public Listener getListener() {
433         throw new UnsupportedOperationException();
434     }
435 }

```

### 6.3 DefaultValidationEventHandler.java

Secuencia 80: javax/xml/bind/helpers/DefaultValidationEventHandler.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.helpers;
27
28 import org.w3c.dom.Node;
29
30 import javax.xml.bind.ValidationEvent;
31 import javax.xml.bind.ValidationEventHandler;
32 import javax.xml.bind.ValidationEventLocator;
33 import java.net.URL;
34
35 /**
36  * <p>
37  * JAXB 1.0 only default validation event handler. This is the default
38  * handler for all objects created from a JAXBContext that is managing
39  * schema-derived code generated by a JAXB 1.0 binding compiler.
40  *
41  * <p>
42  * This handler causes the unmarshal and validate operations to fail on the first
43  * error or fatal error.
44  *
45  * <p>
46  * This handler is not the default handler for JAXB mapped classes following
47  * JAXB 2.0 or later versions. Default validation event handling has changed
48  * and is specified in {@link javax.xml.bind.Unmarshaler} and
49  * {@link javax.xml.bind.Marshaller}.
50  *
51  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
52  * @see javax.xml.bind.Unmarshaler
53  * @see javax.xml.bind.Validator
54  * @see javax.xml.bind.ValidationEventHandler
55  * @since JAXB1.0
56  */
57 public class DefaultValidationEventHandler implements ValidationEventHandler {
58
59     public boolean handleEvent( ValidationEvent event ) {
60
61         if( event == null ) {
```

```
62         throw new IllegalArgumentException();
63     }
64
65     // calculate the severity prefix and return value
66     String severity = null;
67     boolean retVal = false;
68     switch ( event.getSeverity() ) {
69         case ValidationEvent.WARNING:
70             severity = Messages.format( Messages.WARNING );
71             retVal = true; // continue after warnings
72             break;
73         case ValidationEvent.ERROR:
74             severity = Messages.format( Messages.ERROR );
75             retVal = false; // terminate after errors
76             break;
77         case ValidationEvent.FATAL_ERROR:
78             severity = Messages.format( Messages.FATAL_ERROR );
79             retVal = false; // terminate after fatal errors
80             break;
81         default:
82             assert false :
83                 Messages.format( Messages.UNRECOGNIZED_SEVERITY,
84                     event.getSeverity() );
85     }
86
87     // calculate the location message
88     String location = getLocation( event );
89
90     System.out.println(
91         Messages.format( Messages.SEVERITY_MESSAGE,
92             severity,
93             event.getMessage(),
94             location ) );
95
96     // fail on the first error or fatal error
97     return retVal;
98 }
99
100 /**
101  * Calculate a location message for the event
102  *
103  */
104 private String getLocation(ValidationEvent event) {
105     StringBuffer msg = new StringBuffer();
106
107     ValidationEventLocator locator = event.getLocator();
108
109     if( locator != null ) {
110
111         URL url = locator.getURL();
112         Object obj = locator.getObject();
113         Node node = locator.getNode();
114         int line = locator.getLineNumber();
```

```
115
116         if( url!=null || line!=-1 ) {
117             msg.append( "line " + line );
118             if( url!=null )
119                 msg.append( " of " + url );
120         } else if( obj != null ) {
121             msg.append( " obj: " + obj.toString() );
122         } else if( node != null ) {
123             msg.append( " node: " + node.toString() );
124         }
125     } else {
126         msg.append( Messages.format( Messages.LOCATION_UNAVAILABLE ) );
127     }
128
129     return msg.toString();
130 }
131 }
```

## 6.4 Messages.java

Secuencia 81: javax/xml/bind/helpers/Messages.java

```
1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.helpers;
27
28 import java.text.MessageFormat;
29 import java.util.ResourceBundle;
30
31 /**
32  * Formats error messages.
```

```
33  */
34  class Messages
35  {
36      static String format( String property ) {
37          return format( property, null );
38      }
39
40      static String format( String property, Object arg1 ) {
41          return format( property, new Object[]{arg1} );
42      }
43
44      static String format( String property, Object arg1, Object arg2 ) {
45          return format( property, new Object[]{arg1,arg2} );
46      }
47
48      static String format( String property, Object arg1, Object arg2, Object arg3 ) {
49          return format( property, new Object[]{arg1,arg2,arg3} );
50      }
51
52      // add more if necessary.
53
54      /** Loads a string resource and formats it with specified arguments. */
55      static String format( String property, Object[] args ) {
56          String text = ResourceBundle.getBundle(Messages.class.getName()).getString(property);
57          return MessageFormat.format(text,args);
58      }
59
60      //
61      //
62      // Message resources
63      //
64      //
65      static final String INPUTSTREAM_NOT_NULL = // 0 args
66          "AbstractUnmarshallerImpl.IsNotNull";
67
68      static final String MUST_BE_BOOLEAN = // 1 arg
69          "AbstractMarshallerImpl.MustBeBoolean";
70
71      static final String MUST_BE_STRING = // 1 arg
72          "AbstractMarshallerImpl.MustBeString";
73
74      static final String SEVERITY_MESSAGE = // 3 args
75          "DefaultValidationEventHandler.SeverityMessage";
76
77      static final String LOCATION_UNAVAILABLE = // 0 args
78          "DefaultValidationEventHandler.LocationUnavailable";
79
80      static final String UNRECOGNIZED_SEVERITY = // 1 arg
81          "DefaultValidationEventHandler.UnrecognizedSeverity";
82
83      static final String WARNING = // 0 args
84          "DefaultValidationEventHandler.Warning";
85
```

```

86     static final String ERROR = // 0 args
87         "DefaultValidationEventHandler.Error";
88
89     static final String FATAL_ERROR = // 0 args
90         "DefaultValidationEventHandler.FatalError";
91
92     static final String ILLEGAL_SEVERITY = // 0 args
93         "ValidationEventImpl.IllegalSeverity";
94
95     static final String MUST_NOT_BE_NULL = // 1 arg
96         "Shared.MustNotBeNull";
97 }

```

## 6.5 NotIdentifiableEventImpl.java

Secuencia 82: javax/xml/bind/helpers/NotIdentifiableEventImpl.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.helpers;
27
28 import javax.xml.bind.ValidationEventLocator;
29
30 /**
31  * Default implementation of the NotIdentifiableEvent interface.
32  *
33  * <p>
34  * JAXB providers are allowed to use whatever class that implements
35  * the ValidationEvent interface. This class is just provided for a
36  * convenience.
37  *

```

```

38  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
39  * @see javax.xml.bind.NotIdentifiableEvent
40  * @see javax.xml.bind.Validator
41  * @see javax.xml.bind.ValidationEventHandler
42  * @see javax.xml.bind.ValidationEvent
43  * @see javax.xml.bind.ValidationEventLocator
44  * @since JAXB1.0
45  */
46  public class NotIdentifiableEventImpl
47      extends ValidationEventImpl
48      implements javax.xml.bind.NotIdentifiableEvent {
49
50      /**
51       * Create a new NotIdentifiableEventImpl.
52       *
53       * @param _severity The severity value for this event. Must be one of
54       * ValidationEvent.WARNING, ValidationEvent.ERROR, or
55       * ValidationEvent.FATAL_ERROR
56       * @param _message The text message for this event - may be null.
57       * @param _locator The locator object for this event - may be null.
58       * @throws IllegalArgumentException if an illegal severity field is supplied
59       */
60      public NotIdentifiableEventImpl( int _severity, String _message,
61                                       ValidationEventLocator _locator) {
62
63          super(_severity, _message, _locator);
64      }
65
66      /**
67       * Create a new NotIdentifiableEventImpl.
68       *
69       * @param _severity The severity value for this event. Must be one of
70       * ValidationEvent.WARNING, ValidationEvent.ERROR, or
71       * ValidationEvent.FATAL_ERROR
72       * @param _message The text message for this event - may be null.
73       * @param _locator The locator object for this event - may be null.
74       * @param _linkedException An optional linked exception that may provide
75       * additional information about the event - may be null.
76       * @throws IllegalArgumentException if an illegal severity field is supplied
77       */
78      public NotIdentifiableEventImpl( int _severity, String _message,
79                                       ValidationEventLocator _locator,
80                                       Throwable _linkedException) {
81
82          super(_severity, _message, _locator, _linkedException);
83      }
84
85  }

```

## 6.6 ParseConversionEventImpl.java

Secuencia 83: javax/xml/bind/helpers/ParseConversionEventImpl.java

```

1  /*

```



```
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.helpers;
27
28 import javax.xml.bind.ParseConversionEvent;
29 import javax.xml.bind.ValidationEventLocator;
30
31 /**
32  * Default implementation of the ParseConversionEvent interface.
33  *
34  * <p>
35  * JAXB providers are allowed to use whatever class that implements
36  * the ValidationEvent interface. This class is just provided for a
37  * convenience.
38  *
39  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
40  * @see javax.xml.bind.ParseConversionEvent
41  * @see javax.xml.bind.Validator
42  * @see javax.xml.bind.ValidationEventHandler
43  * @see javax.xml.bind.ValidationEvent
44  * @see javax.xml.bind.ValidationEventLocator
45  * @since JAXB1.0
46  */
47 public class ParseConversionEventImpl
48     extends ValidationEventImpl
49     implements ParseConversionEvent {
50
51     /**
52      * Create a new ParseConversionEventImpl.
53      *
54      * @param _severity The severity value for this event. Must be one of
```

```

55     * ValidationEvent.WARNING, ValidationEvent.ERROR, or
56     * ValidationEvent.FATAL_ERROR
57     * @param _message The text message for this event - may be null.
58     * @param _locator The locator object for this event - may be null.
59     * @throws IllegalArgumentException if an illegal severity field is supplied
60     */
61     public ParseConversionEventImpl( int _severity, String _message,
62                                     ValidationEventLocator _locator) {
63
64         super(_severity, _message, _locator);
65     }
66
67     /**
68     * Create a new ParseConversionEventImpl.
69     *
70     * @param _severity The severity value for this event. Must be one of
71     * ValidationEvent.WARNING, ValidationEvent.ERROR, or
72     * ValidationEvent.FATAL_ERROR
73     * @param _message The text message for this event - may be null.
74     * @param _locator The locator object for this event - may be null.
75     * @param _linkedException An optional linked exception that may provide
76     * additional information about the event - may be null.
77     * @throws IllegalArgumentException if an illegal severity field is supplied
78     */
79     public ParseConversionEventImpl( int _severity, String _message,
80                                     ValidationEventLocator _locator,
81                                     Throwable _linkedException) {
82
83         super(_severity, _message, _locator, _linkedException);
84     }
85
86 }

```

## 6.7 PrintConversionEventImpl.java

Secuencia 84: javax/xml/bind/helpers/PrintConversionEventImpl.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.helpers;
27
28 import javax.xml.bind.PrintConversionEvent;
29 import javax.xml.bind.ValidationEventLocator;
30
31 /**
32  * Default implementation of the PrintConversionEvent interface.
33  *
34  * <p>
35  * JAXB providers are allowed to use whatever class that implements
36  * the ValidationEvent interface. This class is just provided for a
37  * convenience.
38  *
39  * @author <ul><li>Ryan Shoemaker, Sun Microsystems, Inc.</li></ul>
40  * @see javax.xml.bind.PrintConversionEvent
41  * @see javax.xml.bind.Validator
42  * @see javax.xml.bind.ValidationEventHandler
43  * @see javax.xml.bind.ValidationEvent
44  * @see javax.xml.bind.ValidationEventLocator
45  * @since JAXB1.0
46  */
47 public class PrintConversionEventImpl
48     extends ValidationEventImpl
49     implements PrintConversionEvent {
50
51     /**
52      * Create a new PrintConversionEventImpl.
53      *
54      * @param _severity The severity value for this event. Must be one of
55      * ValidationEvent.WARNING, ValidationEvent.ERROR, or
56      * ValidationEvent.FATAL_ERROR
57      * @param _message The text message for this event - may be null.
58      * @param _locator The locator object for this event - may be null.
59      * @throws IllegalArgumentException if an illegal severity field is supplied
60      */
61     public PrintConversionEventImpl( int _severity, String _message,
62                                     ValidationEventLocator _locator) {
63
64         super(_severity, _message, _locator);
65     }
66
67     /**
68      * Create a new PrintConversionEventImpl.
69      *
70      * @param _severity The severity value for this event. Must be one of
```

```

71     * ValidationEvent.WARNING, ValidationEvent.ERROR, or
72     * ValidationEvent.FATAL_ERROR
73     * @param _message The text message for this event - may be null.
74     * @param _locator The locator object for this event - may be null.
75     * @param _linkedException An optional linked exception that may provide
76     * additional information about the event - may be null.
77     * @throws IllegalArgumentException if an illegal severity field is supplied
78     */
79     public PrintConversionEventImpl( int _severity, String _message,
80                                     ValidationEventLocator _locator,
81                                     Throwable _linkedException) {
82
83         super(_severity, _message, _locator, _linkedException);
84     }
85
86 }

```

## 6.8 ValidationEventImpl.java

Secuencia 85: javax/xml/bind/helpers/ValidationEventImpl.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.helpers;
27
28 import java.text.MessageFormat;
29
30 import javax.xml.bind.ValidationEvent;
31 import javax.xml.bind.ValidationEventLocator;
32
33 /**

```

```

34  * Default implementation of the ValidationEvent interface.
35  *
36  * <p>
37  * JAXB providers are allowed to use whatever class that implements
38  * the ValidationEvent interface. This class is just provided for a
39  * convenience.
40  *
41  * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li></ul>
42  * @see javax.xml.bind.Validator
43  * @see javax.xml.bind.ValidationEventHandler
44  * @see javax.xml.bind.ValidationEvent
45  * @see javax.xml.bind.ValidationEventLocator
46  * @since JAXB1.0
47  */
48  public class ValidationEventImpl implements ValidationEvent
49  {
50
51      /**
52       * Create a new ValidationEventImpl.
53       *
54       * @param _severity The severity value for this event. Must be one of
55       * ValidationEvent.WARNING, ValidationEvent.ERROR, or
56       * ValidationEvent.FATAL_ERROR
57       * @param _message The text message for this event - may be null.
58       * @param _locator The locator object for this event - may be null.
59       * @throws IllegalArgumentException if an illegal severity field is supplied
60       */
61      public ValidationEventImpl( int _severity, String _message,
62                                ValidationEventLocator _locator ) {
63
64          this(_severity,_message,_locator,null);
65      }
66
67      /**
68       * Create a new ValidationEventImpl.
69       *
70       * @param _severity The severity value for this event. Must be one of
71       * ValidationEvent.WARNING, ValidationEvent.ERROR, or
72       * ValidationEvent.FATAL_ERROR
73       * @param _message The text message for this event - may be null.
74       * @param _locator The locator object for this event - may be null.
75       * @param _linkedException An optional linked exception that may provide
76       * additional information about the event - may be null.
77       * @throws IllegalArgumentException if an illegal severity field is supplied
78       */
79      public ValidationEventImpl( int _severity, String _message,
80                                ValidationEventLocator _locator,
81                                Throwable _linkedException ) {
82
83          setSeverity( _severity );
84          this.message = _message;
85          this.locator = _locator;
86          this.linkedException = _linkedException;

```

```
87     }
88
89     private int severity;
90     private String message;
91     private Throwable linkedException;
92     private ValidationEventLocator locator;
93
94     public int getSeverity() {
95         return severity;
96     }
97
98
99     /**
100      * Set the severity field of this event.
101      *
102      * @param _severity Must be one of ValidationEvent.WARNING,
103      * ValidationEvent.ERROR, or ValidationEvent.FATAL_ERROR.
104      * @throws IllegalArgumentException if an illegal severity field is supplied
105      */
106     public void setSeverity( int _severity ) {
107
108         if( _severity != ValidationEvent.WARNING &&
109             _severity != ValidationEvent.ERROR &&
110             _severity != ValidationEvent.FATAL_ERROR ) {
111             throw new IllegalArgumentException(
112                 Messages.format( Messages.ILLEGAL_SEVERITY ) );
113         }
114
115         this.severity = _severity;
116     }
117
118     public String getMessage() {
119         return message;
120     }
121
122     /**
123      * Set the message field of this event.
124      *
125      * @param _message String message - may be null.
126      */
127     public void setMessage( String _message ) {
128         this.message = _message;
129     }
130
131     public Throwable getLinkedException() {
132         return linkedException;
133     }
134
135     /**
136      * Set the linked exception field of this event.
137      *
138      * @param _linkedException Optional linked exception - may be null.
139      */
140     public void setLinkedException( Throwable _linkedException ) {
141         this.linkedException = _linkedException;
142     }
143 }
```

```

140     }
141
142     public ValidationEventLocator getLocator() {
143         return locator;
144     }
145     /**
146      * Set the locator object for this event.
147      *
148      * @param _locator The locator - may be null.
149      */
150     public void setLocator( ValidationEventLocator _locator ) {
151         this.locator = _locator;
152     }
153
154     /**
155      * Returns a string representation of this object in a format
156      * helpful to debugging.
157      *
158      * @see Object#equals(Object)
159      */
160     public String toString() {
161         String s;
162         switch(getSeverity()) {
163             case WARNING: s="WARNING";break;
164             case ERROR: s="ERROR";break;
165             case FATAL_ERROR: s="FATAL_ERROR";break;
166             default: s=String.valueOf(getSeverity());break;
167         }
168         return MessageFormat.format("[severity={0},message={1},locator={2}]",
169             new Object[]{
170                 s,
171                 getMessage(),
172                 getLocator()
173             });
174     }
175 }

```

## 6.9 ValidationEventLocatorImpl.java

Secuencia 86: javax/xml/bind/helpers/ValidationEventLocatorImpl.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.bind.helpers;
27
28 import java.net.URL;
29 import java.net.MalformedURLException;
30 import java.text.MessageFormat;
31
32 import javax.xml.bind.ValidationEventLocator;
33 import org.w3c.dom.Node;
34 import org.xml.sax.Locator;
35 import org.xml.sax.SAXParseException;
36
37 /**
38  * Default implementation of the ValidationEventLocator interface.
39  *
40  * <p>
41  * JAXB providers are allowed to use whatever class that implements
42  * the ValidationEventLocator interface. This class is just provided for a
43  * convenience.
44  *
45  * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li></ul>
46  * @see javax.xml.bind.Validator
47  * @see javax.xml.bind.ValidationEventHandler
48  * @see javax.xml.bind.ValidationEvent
49  * @see javax.xml.bind.ValidationEventLocator
50  * @since JAXB1.0
51  */
52 public class ValidationEventLocatorImpl implements ValidationEventLocator
53 {
54     /**
55      * Creates an object with all fields unavailable.
56      */
57     public ValidationEventLocatorImpl() {
58     }
59
60     /**
61      * Constructs an object from an org.xml.sax.Locator.
62      *
63      * The object's ColumnNumber, LineNumber, and URL become available from the
64      * values returned by the locator's getColumnNumber(), getLineNumber(), and
65      * getSystemId() methods respectively. Node, Object, and Offset are not
66      * available.
```



```

67      *
68      * @param loc the SAX Locator object that will be used to populate this
69      * event locator.
70      * @throws IllegalArgumentException if the Locator is null
71      */
72      public ValidationEventLocatorImpl( Locator loc ) {
73          if( loc == null ) {
74              throw new IllegalArgumentException(
75                  Messages.format( Messages.MUST_NOT_BE_NULL, "loc" ) );
76          }
77
78          this.url = toURL(loc.getSystemId());
79          this.columnNumber = loc.getColumnNumber();
80          this.lineNumber = loc.getLineNumber();
81      }
82
83      /**
84       * Constructs an object from the location information of a SAXParseException.
85       *
86       * The object's ColumnNumber, LineNumber, and URL become available from the
87       * values returned by the locator's getColumnNumber(), getLineNumber(), and
88       * getSystemId() methods respectively. Node, Object, and Offset are not
89       * available.
90       *
91       * @param e the SAXParseException object that will be used to populate this
92       * event locator.
93       * @throws IllegalArgumentException if the SAXParseException is null
94       */
95      public ValidationEventLocatorImpl( SAXParseException e ) {
96          if( e == null ) {
97              throw new IllegalArgumentException(
98                  Messages.format( Messages.MUST_NOT_BE_NULL, "e" ) );
99          }
100
101          this.url = toURL(e.getSystemId());
102          this.columnNumber = e.getColumnNumber();
103          this.lineNumber = e.getLineNumber();
104      }
105
106      /**
107       * Constructs an object that points to a DOM Node.
108       *
109       * The object's Node becomes available. ColumnNumber, LineNumber, Object,
110       * Offset, and URL are not available.
111       *
112       * @param _node the DOM Node object that will be used to populate this
113       * event locator.
114       * @throws IllegalArgumentException if the Node is null
115       */
116      public ValidationEventLocatorImpl(Node _node) {
117          if( _node == null ) {
118              throw new IllegalArgumentException(
119                  Messages.format( Messages.MUST_NOT_BE_NULL, "_node" ) );

```

```

120     }
121
122     this.node = _node;
123 }
124
125 /**
126  * Constructs an object that points to a JAXB content object.
127  *
128  * The object's Object becomes available. ColumnNumber, LineNumber, Node,
129  * Offset, and URL are not available.
130  *
131  * @param _object the Object that will be used to populate this
132  * event locator.
133  * @throws IllegalArgumentException if the Object is null
134  */
135 public ValidationEventLocatorImpl(Object _object) {
136     if( _object == null ) {
137         throw new IllegalArgumentException(
138             Messages.format( Messages.MUST_NOT_BE_NULL, "_object" ) );
139     }
140
141     this.object = _object;
142 }
143
144 /** Converts a system ID to an URL object. */
145 private static URL toURL( String systemId ) {
146     try {
147         return new URL(systemId);
148     } catch( MalformedURLException e ) {
149         // TODO: how should we handle system id here?
150         return null;    // for now
151     }
152 }
153
154 private URL url = null;
155 private int offset = -1;
156 private int lineNumber = -1;
157 private int columnNumber = -1;
158 private Object object = null;
159 private Node node = null;
160
161
162 /**
163  * @see javax.xml.bind.ValidationEventLocator#getURL()
164  */
165 public URL getURL() {
166     return url;
167 }
168
169 /**
170  * Set the URL field on this event locator. Null values are allowed.
171  *
172  * @param _url the url

```

```
173     */
174     public void setURL( URL _url ) {
175         this.url = _url;
176     }
177
178     /**
179     * @see javax.xml.bind.ValidationEventLocator#getOffset()
180     */
181     public int getOffset() {
182         return offset;
183     }
184
185     /**
186     * Set the offset field on this event locator.
187     *
188     * @param _offset the offset
189     */
190     public void setOffset( int _offset ) {
191         this.offset = _offset;
192     }
193
194     /**
195     * @see javax.xml.bind.ValidationEventLocator#getLineNumber()
196     */
197     public int getLineNumber() {
198         return lineNumber;
199     }
200
201     /**
202     * Set the lineNumber field on this event locator.
203     *
204     * @param _lineNumber the line number
205     */
206     public void setLineNumber( int _lineNumber ) {
207         this.lineNumber = _lineNumber;
208     }
209
210     /**
211     * @see javax.xml.bind.ValidationEventLocator#getColumnNumber()
212     */
213     public int getColumnNumber() {
214         return columnNumber;
215     }
216
217     /**
218     * Set the columnNumber field on this event locator.
219     *
220     * @param _columnNumber the column number
221     */
222     public void setColumnNumber( int _columnNumber ) {
223         this.columnNumber = _columnNumber;
224     }
225
```

```

226  /**
227   * @see javax.xml.bind.ValidationEventLocator#getObject()
228   */
229  public Object getObject() {
230      return object;
231  }
232
233  /**
234   * Set the Object field on this event locator. Null values are allowed.
235   *
236   * @param _object the java content object
237   */
238  public void setObject( Object _object ) {
239      this.object = _object;
240  }
241
242  /**
243   * @see javax.xml.bind.ValidationEventLocator#getNode()
244   */
245  public Node getNode() {
246      return node;
247  }
248
249  /**
250   * Set the Node field on this event locator. Null values are allowed.
251   *
252   * @param _node the Node
253   */
254  public void setNode( Node _node ) {
255      this.node = _node;
256  }
257
258  /**
259   * Returns a string representation of this object in a format
260   * helpful to debugging.
261   *
262   * @see Object#equals(Object)
263   */
264  public String toString() {
265      return MessageFormat.format("[node={0},object={1},url={2},line={3},col={4},offset={5}]",
266          getNode(),
267          getObject(),
268          getURL(),
269          String.valueOf(getLineNumber()),
270          String.valueOf(getColumnNumber()),
271          String.valueOf(getOffset()));
272  }
273 }

```

## 7 Xml Bind Util (javax.xml.bind.util package)

### 7.1 JAXBResult.java

Secuencia 87: javax/xml/bind/util/JAXBResult.java

```
1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.util;
27
28 import javax.xml.bind.JAXBContext;
29 import javax.xml.bind.JAXBException;
30 import javax.xml.bind.Unmarshaller;
31 import javax.xml.bind.UnmarshallerHandler;
32 import javax.xml.transform.sax.SAXResult;
33
34 /**
35  * JAXP {@link javax.xml.transform.Result} implementation
36  * that unmarshals a JAXB object.
37  *
38  * <p>
39  * This utility class is useful to combine JAXB with
40  * other Java/XML technologies.
41  *
42  * <p>
43  * The following example shows how to use JAXB to unmarshal a document
44  * resulting from an XSLT transformation.
45  *
46  * <blockquote>
47  *   <pre>
48  *       JAXBResult result = new JAXBResult(
49  *           JAXBContext.newInstance("org.acme.foo" ) );
50  *
51  *       // set up XSLT transformation
52  *       TransformerFactory tf = TransformerFactory.newInstance();
```

```

53 *      Transformer t = tf.newTransformer(new StreamSource("test.xml"));
54 *
55 *      // run transformation
56 *      t.transform(new StreamSource("document.xml"),result);
57 *
58 *      // obtain the unmarshalled content tree
59 *      Object o = result.getResult();
60 * </pre>
61 * </blockquote>
62 *
63 * <p>
64 * The fact that JAXBResult derives from SAXResult is an implementation
65 * detail. Thus in general applications are strongly discouraged from
66 * accessing methods defined on SAXResult.
67 *
68 * <p>
69 * In particular it shall never attempt to call the setHandler,
70 * setLexicalHandler, and setSystemId methods.
71 *
72 * @author
73 *      Kohsuke Kawaguchi (kohsuke.kawaguchi@sun.com)
74 */
75 public class JAXBResult extends SAXResult {
76
77     /**
78      * Creates a new instance that uses the specified
79      * JAXBContext to unmarshal.
80      *
81      * @param context The JAXBContext that will be used to create the
82      * necessary Unmarshaller. This parameter must not be null.
83      * @exception JAXBException if an error is encountered while creating the
84      * JAXBResult or if the context parameter is null.
85      */
86     public JAXBResult( JAXBContext context ) throws JAXBException {
87         this( ( context == null ) ? assertionFailed() : context.createUnmarshaller() );
88     }
89
90     /**
91      * Creates a new instance that uses the specified
92      * Unmarshaller to unmarshal an object.
93      *
94      * <p>
95      * This JAXBResult object will use the specified Unmarshaller
96      * instance. It is the caller's responsibility not to use the
97      * same Unmarshaller for other purposes while it is being
98      * used by this object.
99      *
100     * <p>
101     * The primary purpose of this method is to allow the client
102     * to configure Unmarshaller. Unless you know what you are doing,
103     * it's easier and safer to pass a JAXBContext.
104     *
105     * @param _unmarshaller the unmarshaller. This parameter must not be null.

```

```

106     * @throws JAXBException if an error is encountered while creating the
107     * JAXBResult or the Unmarshaller parameter is null.
108     */
109     public JAXBResult( Unmarshaller _unmarshaller ) throws JAXBException {
110         if( _unmarshaller == null )
111             throw new JAXBException(
112                 Messages.format( Messages.RESULT_NULL_UNMARSHALLER ) );
113
114         this.unmarshallerHandler = _unmarshaller.getUnmarshallerHandler();
115
116         super.setHandler(unmarshallerHandler);
117     }
118
119     /**
120     * Unmarshaller that will be used to unmarshal
121     * the input documents.
122     */
123     private final UnmarshallerHandler unmarshallerHandler;
124
125     /**
126     * Gets the unmarshalled object created by the transformation.
127     *
128     * @return
129     *     Always return a non-null object.
130     *
131     * @exception IllegalStateException
132     *     if this method is called before an object is unmarshalled.
133     *
134     * @exception JAXBException
135     *     if there is any unmarshalling error.
136     *     Note that the implementation is allowed to throw SAXException
137     *     during the parsing when it finds an error.
138     */
139     public Object getResult() throws JAXBException {
140         return unmarshallerHandler.getResult();
141     }
142
143     /**
144     * Hook to throw exception from the middle of a constructor chained call
145     * to this
146     */
147     private static Unmarshaller assertionFailed() throws JAXBException {
148         throw new JAXBException( Messages.format( Messages.RESULT_NULL_CONTEXT ) );
149     }
150 }

```

## 7.2 JAXBSource.java

Secuencia 88: javax/xml/bind/util/JAXBSource.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *

```

```
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.util;
27
28 import org.xml.sax.ContentHandler;
29 import org.xml.sax.DTDHandler;
30 import org.xml.sax.EntityResolver;
31 import org.xml.sax.ErrorHandler;
32 import org.xml.sax.InputSource;
33 import org.xml.sax.SAXException;
34 import org.xml.sax.SAXNotRecognizedException;
35 import org.xml.sax.SAXParseException;
36 import org.xml.sax.XMLReader;
37 import org.xml.sax.ext.LexicalHandler;
38 import org.xml.sax.helpers.XMLFilterImpl;
39
40 import javax.xml.bind.JAXBContext;
41 import javax.xml.bind.JAXBException;
42 import javax.xml.bind.Marshaller;
43 import javax.xml.transform.sax.SAXSource;
44 import org.xml.sax.XMLFilter;
45
46 /**
47  * JAXP {@link javax.xml.transform.Source} implementation
48  * that marshals a JAXB-generated object.
49  *
50  * <p>
51  * This utility class is useful to combine JAXB with
52  * other Java/XML technologies.
53  *
54  * <p>
55  * The following example shows how to use JAXB to marshal a document
56  * for transformation by XSLT.
57  *
```



```

58 * <blockquote>
59 *   <pre>
60 *       MyObject o = // get JAXB content tree
61 *
62 *       // jaxbContext is a JAXBContext object from which 'o' is created.
63 *       JAXBSource source = new JAXBSource( jaxbContext, o );
64 *
65 *       // set up XSLT transformation
66 *       TransformerFactory tf = TransformerFactory.newInstance();
67 *       Transformer t = tf.newTransformer(new StreamSource("test.xsl"));
68 *
69 *       // run transformation
70 *       t.transform(source,new StreamResult(System.out));
71 *   </pre>
72 * </blockquote>
73 *
74 * <p>
75 * The fact that JAXBSource derives from SAXSource is an implementation
76 * detail. Thus in general applications are strongly discouraged from
77 * accessing methods defined on SAXSource. In particular,
78 * the setXMLReader and setInputSource methods shall never be called.
79 * The XMLReader object obtained by the getXMLReader method shall
80 * be used only for parsing the InputSource object returned by
81 * the getInputSource method.
82 *
83 * <p>
84 * Similarly the InputSource object obtained by the getInputSource
85 * method shall be used only for being parsed by the XMLReader object
86 * returned by the getXMLReader.
87 *
88 * @author
89 *     Kohsuke Kawaguchi (kohsuke.kawaguchi@sun.com)
90 */
91 public class JAXBSource extends SAXSource {
92
93     /**
94      * Creates a new {@link javax.xml.transform.Source} for the given content object.
95      *
96      * @param context
97      *     JAXBContext that was used to create
98      *     <code>contentObject</code>. This context is used
99      *     to create a new instance of marshaller and must not be null.
100     * @param contentObject
101     *     An instance of a JAXB-generated class, which will be
102     *     used as a {@link javax.xml.transform.Source} (by marshalling it into XML). It must
103     *     not be null.
104     * @throws JAXBException if an error is encountered while creating the
105     *     JAXBSource or if either of the parameters are null.
106     */
107     public JAXBSource( JAXBContext context, Object contentObject )
108         throws JAXBException {
109
110         this(

```

```

111         ( context == null ) ?
112             assertionFailed( Messages.format( Messages.SOURCE_NULL_CONTEXT ) ) :
113             context.createMarshaller(),
114
115         ( contentObject == null ) ?
116             assertionFailed( Messages.format( Messages.SOURCE_NULL_CONTENT ) ) :
117             contentObject);
118     }
119
120     /**
121     * Creates a new {@link javax.xml.transform.Source} for the given content object.
122     *
123     * @param marshaller
124     *     A marshaller instance that will be used to marshal
125     *     <code>contentObject</code> into XML. This must be
126     *     created from a JAXBContext that was used to build
127     *     <code>contentObject</code> and must not be null.
128     * @param contentObject
129     *     An instance of a JAXB-generated class, which will be
130     *     used as a {@link javax.xml.transform.Source} (by marshalling it into XML). It must
131     *     not be null.
132     * @throws JAXBException if an error is encountered while creating the
133     *     JAXBSource or if either of the parameters are null.
134     */
135     public JAXBSource( Marshaller marshaller, Object contentObject )
136         throws JAXBException {
137
138         if( marshaller == null )
139             throw new JAXBException(
140                 Messages.format( Messages.SOURCE_NULL_MARSHALLER ) );
141
142         if( contentObject == null )
143             throw new JAXBException(
144                 Messages.format( Messages.SOURCE_NULL_CONTENT ) );
145
146         this.marshaller = marshaller;
147         this.contentObject = contentObject;
148
149         super.setXMLReader(pseudoParser);
150         // pass a dummy InputSource. We don't care
151         super.setInputSource(new InputSource());
152     }
153
154     private final Marshaller marshaller;
155     private final Object contentObject;
156
157     // this object will pretend as an XMLReader.
158     // no matter what parameter is specified to the parse method,
159     // it just parse the contentObject.
160     private final XMLReader pseudoParser = new XMLReader() {
161         public boolean getFeature(String name) throws SAXNotRecognizedException {
162             if(name.equals("http://xml.org/sax/features/namespaces"))
163                 return true;

```

```
164         if(name.equals("http://xml.org/sax/features/namespace-prefixes"))
165             return false;
166         throw new SAXNotRecognizedException(name);
167     }
168
169     public void setFeature(String name, boolean value) throws SAXNotRecognizedException {
170         if(name.equals("http://xml.org/sax/features/namespace-prefixes") && value)
171             return;
172         if(name.equals("http://xml.org/sax/features/namespace-prefixes") && !value)
173             return;
174         throw new SAXNotRecognizedException(name);
175     }
176
177     public Object getProperty(String name) throws SAXNotRecognizedException {
178         if( "http://xml.org/sax/properties/lexical-handler".equals(name) ) {
179             return lexicalHandler;
180         }
181         throw new SAXNotRecognizedException(name);
182     }
183
184     public void setProperty(String name, Object value) throws SAXNotRecognizedException {
185         if( "http://xml.org/sax/properties/lexical-handler".equals(name) ) {
186             this.lexicalHandler = (LexicalHandler)value;
187             return;
188         }
189         throw new SAXNotRecognizedException(name);
190     }
191
192     private LexicalHandler lexicalHandler;
193
194     // we will store this value but never use it by ourselves.
195     private EntityResolver entityResolver;
196     public void setEntityResolver(EntityResolver resolver) {
197         this.entityResolver = resolver;
198     }
199     public EntityResolver getEntityResolver() {
200         return entityResolver;
201     }
202
203     private DTDHandler dtdHandler;
204     public void setDTDHandler(DTDHandler handler) {
205         this.dtdHandler = handler;
206     }
207     public DTDHandler getDTDHandler() {
208         return dtdHandler;
209     }
210
211     // SAX allows ContentHandler to be changed during the parsing,
212     // but JAXB doesn't. So this repeater will sit between those
213     // two components.
214     private XMLFilter repeater = new XMLFilterImpl();
215
216     public void setContentHandler(ContentHandler handler) {
```

```

217     repeater.setContentHandler(handler);
218 }
219 public ContentHandler getContentHandler() {
220     return repeater.getContentHandler();
221 }
222
223 private ErrorHandler errorHandler;
224 public void setErrorHandler(ErrorHandler handler) {
225     this.errorHandler = handler;
226 }
227 public ErrorHandler getErrorHandler() {
228     return errorHandler;
229 }
230
231 public void parse(InputSource input) throws SAXException {
232     parse();
233 }
234
235 public void parse(String systemId) throws SAXException {
236     parse();
237 }
238
239 public void parse() throws SAXException {
240     // parses a content object by using the given marshaller
241     // SAX events will be sent to the repeater, and the repeater
242     // will further forward it to an appropriate component.
243     try {
244         marshaller.marshal( contentObject, (XMLFilterImpl)repeater );
245     } catch( JAXBException e ) {
246         // wrap it to a SAXException
247         SAXParseException se =
248             new SAXParseException( e.getMessage(),
249                 null, null, -1, -1, e );
250
251         // if the consumer sets an error handler, it is our responsibility
252         // to notify it.
253         if(errorHandler!=null)
254             errorHandler.fatalError(se);
255
256         // this is a fatal error. Even if the error handler
257         // returns, we will abort anyway.
258         throw se;
259     }
260 }
261 };
262
263 /**
264  * Hook to throw exception from the middle of a constructor chained call
265  * to this
266  */
267 private static Marshaller assertionFailed( String message )
268     throws JAXBException {
269

```

```
270     throw new JAXBException( message );
271 }
272 }
```

### 7.3 Messages.java

Secuencia 89: javax/xml/bind/util/Messages.java

```
1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.bind.util;
27
28 import java.text.MessageFormat;
29 import java.util.ResourceBundle;
30
31 /**
32  * Formats error messages.
33  */
34 class Messages
35 {
36     static String format( String property ) {
37         return format( property, null );
38     }
39
40     static String format( String property, Object arg1 ) {
41         return format( property, new Object[]{arg1} );
42     }
43
44     static String format( String property, Object arg1, Object arg2 ) {
45         return format( property, new Object[]{arg1,arg2} );
46     }
```

```

47
48     static String format( String property, Object arg1, Object arg2, Object arg3 ) {
49         return format( property, new Object[] {arg1,arg2,arg3} );
50     }
51
52     // add more if necessary.
53
54     /** Loads a string resource and formats it with specified arguments. */
55     static String format( String property, Object[] args ) {
56         String text = ResourceBundle.getBundle(Messages.class.getName()).getString(property);
57         return MessageFormat.format(text,args);
58     }
59
60     //
61     //
62     // Message resources
63     //
64     //
65     static final String UNRECOGNIZED_SEVERITY = // 1 arg
66         "ValidationEventCollector.UnrecognizedSeverity";
67
68     static final String RESULT_NULL_CONTEXT = // 0 args
69         "JAXBResult.NullContext";
70
71     static final String RESULT_NULL_UNMARSHALLER = // 0 arg
72         "JAXBResult.NullUnmarshaller";
73
74     static final String SOURCE_NULL_CONTEXT = // 0 args
75         "JAXBSource.NullContext";
76
77     static final String SOURCE_NULL_CONTENT = // 0 arg
78         "JAXBSource.NullContent";
79
80     static final String SOURCE_NULL_MARSHALLER = // 0 arg
81         "JAXBSource.NullMarshaller";
82
83 }

```

## 7.4 ValidationEventCollector.java

Secuencia 90: javax/xml/bind/util/ValidationEventCollector.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```

13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.bind.util;
27
28  import javax.xml.bind.ValidationEvent;
29  import javax.xml.bind.ValidationEventHandler;
30  import java.util.ArrayList;
31  import java.util.List;
32
33  /**
34   * {@link javax.xml.bind.ValidationEventHandler ValidationEventHandler}
35   * implementation that collects all events.
36   *
37   * <p>
38   * To use this class, create a new instance and pass it to the setEventHandler
39   * method of the Validator, Unmarshaller, Marshaller class. After the call to
40   * validate or unmarshal completes, call the getEvents method to retrieve all
41   * the reported errors and warnings.
42   *
43   * @author <ul><li>Kohsuke Kawaguchi, Sun Microsystems, Inc.</li><li>Ryan Shoemaker, Sun
44   * Microsystems, Inc.</li><li>Joe Fialli, Sun Microsystems, Inc.</li></ul>
45   * @see javax.xml.bind.Validator
46   * @see javax.xml.bind.ValidationEventHandler
47   * @see javax.xml.bind.ValidationEvent
48   * @see javax.xml.bind.ValidationEventLocator
49   * @since JAXB1.0
50   */
51  public class ValidationEventCollector implements ValidationEventHandler
52  {
53      private final List<ValidationEvent> events = new ArrayList<ValidationEvent>();
54
55      /**
56       * Return an array of ValidationEvent objects containing a copy of each of
57       * the collected errors and warnings.
58       *
59       * @return
60       *     a copy of all the collected errors and warnings or an empty array
61       *     if there weren't any
62       */
63      public ValidationEvent[] getEvents() {
64          return events.toArray(new ValidationEvent[events.size()]);
65      }

```

```
65
66  /**
67   * Clear all collected errors and warnings.
68   */
69  public void reset() {
70      events.clear();
71  }
72
73  /**
74   * Returns true if this event collector contains at least one
75   * ValidationEvent.
76   *
77   * @return true if this event collector contains at least one
78   *         ValidationEvent, false otherwise
79   */
80  public boolean hasEvents() {
81      return !events.isEmpty();
82  }
83
84  public boolean handleEvent( ValidationEvent event ) {
85      events.add(event);
86
87      boolean retVal = true;
88      switch( event.getSeverity() ) {
89          case ValidationEvent.WARNING:
90              retVal = true; // continue validation
91              break;
92          case ValidationEvent.ERROR:
93              retVal = true; // continue validation
94              break;
95          case ValidationEvent.FATAL_ERROR:
96              retVal = false; // halt validation
97              break;
98          default:
99              _assert( false,
100                      Messages.format( Messages.UNRECOGNIZED_SEVERITY,
101                                      event.getSeverity() ) );
102              break;
103      }
104
105      return retVal;
106  }
107
108  private static void _assert( boolean b, String msg ) {
109      if( !b ) {
110          throw new InternalError( msg );
111      }
112  }
113 }
```



## 8 Xml Crypto (javax.xml.crypto package)

### 8.1 AlgorithmMethod.java

Secuencia 91: javax/xml/crypto/AlgorithmMethod.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: AlgorithmMethod.java,v 1.4 2005/05/10 15:47:41 mullan Exp $
27 */
28 package javax.xml.crypto;
29
30 import java.security.spec.AlgorithmParameterSpec;
31
32 /**
33 * An abstract representation of an algorithm defined in the XML Security
34 * specifications. Subclasses represent specific types of XML security
35 * algorithms, such as a {@link javax.xml.crypto.dsig.Transform}.
36 *
37 * @author Sean Mullan
38 * @author JSR 105 Expert Group
39 * @since 1.6
40 */
41 public interface AlgorithmMethod {
42
43     /**
44      * Returns the algorithm URI of this <code>AlgorithmMethod</code>.
45      *
46      * @return the algorithm URI of this <code>AlgorithmMethod</code>
47      */

```

```

48     String getAlgorithm();
49
50     /**
51      * Returns the algorithm parameters of this <code>AlgorithmMethod</code>.
52      *
53      * @return the algorithm parameters of this <code>AlgorithmMethod</code>.
54      * Returns <code>null</code> if this <code>AlgorithmMethod</code> does
55      * not require parameters and they are not specified.
56      */
57     AlgorithmParameterSpec getParameterSpec();
58 }

```

## 8.2 Data.java

Secuencia 92: javax/xml/crypto/Data.java

```

1  /*
2   * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25 /*
26  * $Id: Data.java,v 1.4 2005/05/10 15:47:41 mullan Exp $
27  */
28 package javax.xml.crypto;
29
30 import javax.xml.crypto.dsig.Transform;
31
32 /**
33  * An abstract representation of the result of dereferencing a
34  * {@link URIReference} or the input/output of subsequent {@link Transform}s.
35  * The primary purpose of this interface is to group and provide type safety
36  * for all <code>Data</code> subtypes.
37  *
38  * @author Sean Mullan

```

```

39  * @author JSR 105 Expert Group
40  * @since 1.6
41  */
42  public interface Data { }

```

### 8.3 KeySelector.java

Secuencia 93: javax/xml/crypto/KeySelector.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: KeySelector.java,v 1.6 2005/05/10 15:47:42 mullan Exp $
27 */
28 package javax.xml.crypto;
29
30 import java.security.Key;
31 import javax.xml.crypto.dsig.keyinfo.KeyInfo;
32 import javax.xml.crypto.dsig.keyinfo.RetrievalMethod;
33
34 /**
35  * A selector that finds and returns a key using the data contained in a
36  * {@link KeyInfo} object. An example of an implementation of
37  * this class is one that searches a {@link java.security.KeyStore} for
38  * trusted keys that match information contained in a KeyInfo.
39  *
40  * <p>Whether or not the returned key is trusted and the mechanisms
41  * used to determine that is implementation-specific.
42  *
43  * @author Sean Mullan
44  * @author JSR 105 Expert Group
45  * @since 1.6

```

```

46  */
47  public abstract class KeySelector {
48
49      /**
50       * The purpose of the key that is to be selected.
51       */
52      public static class Purpose {
53
54          private final String name;
55
56          private Purpose(String name)    { this.name = name; }
57
58          /**
59           * Returns a string representation of this purpose ("sign",
60           * "verify", "encrypt", or "decrypt").
61           *
62           * @return a string representation of this purpose
63           */
64          public String toString()        { return name; }
65
66          /**
67           * A key for signing.
68           */
69          public static final Purpose SIGN = new Purpose("sign");
70          /**
71           * A key for verifying.
72           */
73          public static final Purpose VERIFY = new Purpose("verify");
74          /**
75           * A key for encrypting.
76           */
77          public static final Purpose ENCRYPT = new Purpose("encrypt");
78          /**
79           * A key for decrypting.
80           */
81          public static final Purpose DECRYPT = new Purpose("decrypt");
82      }
83
84      /**
85       * Default no-args constructor; intended for invocation by subclasses only.
86       */
87      protected KeySelector() {}
88
89      /**
90       * Attempts to find a key that satisfies the specified constraints.
91       *
92       * @param keyInfo a KeyInfo (may be null)
93       * @param purpose the key's purpose ({@link Purpose#SIGN},
94       *     {@link Purpose#VERIFY}, {@link Purpose#ENCRYPT}, or
95       *     {@link Purpose#DECRYPT})
96       * @param method the algorithm method that this key is to be used for.
97       *     Only keys that are compatible with the algorithm and meet the
98       *     constraints of the specified algorithm should be returned.

```

```

99      * @param context an XMLCryptoContext that may contain
100     *     useful information for finding an appropriate key. If this key
101     *     selector supports resolving {@link RetrievalMethod} types, the
102     *     context's baseURI and dereferencer
103     *     parameters (if specified) should be used by the selector to
104     *     resolve and dereference the URI.
105     * @return the result of the key selector
106     * @throws KeySelectorException if an exceptional condition occurs while
107     *     attempting to find a key. Note that an inability to find a key is not
108     *     considered an exception (null should be
109     *     returned in that case). However, an error condition (ex: network
110     *     communications failure) that prevented the KeySelector
111     *     from finding a potential key should be considered an exception.
112     * @throws ClassCastException if the data type of method
113     *     is not supported by this key selector
114     */
115     public abstract KeySelectorResult select(KeyInfo keyInfo, Purpose purpose,
116         AlgorithmMethod method, XMLCryptoContext context)
117         throws KeySelectorException;
118
119     /**
120     * Returns a KeySelector that always selects the specified
121     * key, regardless of the KeyInfo passed to it.
122     *
123     * @param key the sole key to be stored in the key selector
124     * @return a key selector that always selects the specified key
125     * @throws NullPointerException if key is null
126     */
127     public static KeySelector singletonKeySelector(Key key) {
128         return new SingletonKeySelector(key);
129     }
130
131     private static class SingletonKeySelector extends KeySelector {
132         private final Key key;
133
134         SingletonKeySelector(Key key) {
135             if (key == null) {
136                 throw new NullPointerException();
137             }
138             this.key = key;
139         }
140
141         public KeySelectorResult select(KeyInfo keyInfo, Purpose purpose,
142             AlgorithmMethod method, XMLCryptoContext context)
143             throws KeySelectorException {
144
145             return new KeySelectorResult() {
146                 public Key getKey() {
147                     return key;
148                 }
149             };
150         }
151     }

```

152 }

## 8.4 KeySelectorException.java

Secuencia 94: javax/xml/crypto/KeySelectorException.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: KeySelectorException.java,v 1.3 2005/05/10 15:47:42 mullan Exp $
27 */
28 package javax.xml.crypto;
29
30 import java.io.PrintStream;
31 import java.io.PrintWriter;
32
33 /**
34 * Indicates an exceptional condition thrown by a {@link KeySelector}.
35 *
36 * <p>A KeySelectorException can contain a cause: another
37 * throwable that caused this KeySelectorException to get thrown.
38 *
39 * @author Sean Mullan
40 * @author JSR 105 Expert Group
41 * @since 1.6
42 */
43 public class KeySelectorException extends Exception {
44
45     private static final long serialVersionUID = -7480033639322531109L;
46
47     /**
48      * The throwable that caused this exception to get thrown, or

```

```

49     * <code>null</code> if this exception was not caused by another throwable
50     * or if the causative throwable is unknown.
51     *
52     * @serial
53     */
54     private Throwable cause;
55
56     /**
57     * Constructs a new <code>KeySelectorException</code> with
58     * <code>null</code> as its detail message.
59     */
60     public KeySelectorException() {
61         super();
62     }
63
64     /**
65     * Constructs a new <code>KeySelectorException</code> with the specified
66     * detail message.
67     *
68     * @param message the detail message
69     */
70     public KeySelectorException(String message) {
71         super(message);
72     }
73
74     /**
75     * Constructs a new <code>KeySelectorException</code> with the
76     * specified detail message and cause.
77     * <p>Note that the detail message associated with
78     * <code>cause</code> is <i>not</i> automatically incorporated in
79     * this exception's detail message.
80     *
81     * @param message the detail message
82     * @param cause the cause (A <tt>null</tt> value is permitted, and
83     *     indicates that the cause is nonexistent or unknown.)
84     */
85     public KeySelectorException(String message, Throwable cause) {
86         super(message);
87         this.cause = cause;
88     }
89
90     /**
91     * Constructs a new <code>KeySelectorException</code> with the specified
92     * cause and a detail message of
93     * <code>(cause==null ? null : cause.toString())</code>
94     * (which typically contains the class and detail message of
95     * <code>cause</code>).
96     *
97     * @param cause the cause (A <tt>null</tt> value is permitted, and
98     *     indicates that the cause is nonexistent or unknown.)
99     */
100     public KeySelectorException(Throwable cause) {
101         super(cause==null ? null : cause.toString());

```

```

102     this.cause = cause;
103 }
104
105 /**
106  * Returns the cause of this <code>KeySelectorException</code> or
107  * <code>>null</code> if the cause is nonexistent or unknown. (The
108  * cause is the throwable that caused this
109  * <code>KeySelectorException</code> to get thrown.)
110  *
111  * @return the cause of this <code>KeySelectorException</code> or
112  *         <code>>null</code> if the cause is nonexistent or unknown.
113  */
114 public Throwable getCause() {
115     return cause;
116 }
117
118 /**
119  * Prints this <code>KeySelectorException</code>, its backtrace and
120  * the cause's backtrace to the standard error stream.
121  */
122 public void printStackTrace() {
123     super.printStackTrace();
124     //XXX print backtrace of cause
125 }
126
127 /**
128  * Prints this <code>KeySelectorException</code>, its backtrace and
129  * the cause's backtrace to the specified print stream.
130  *
131  * @param s <code>PrintStream</code> to use for output
132  */
133 public void printStackTrace(PrintStream s) {
134     super.printStackTrace(s);
135     //XXX print backtrace of cause
136 }
137
138 /**
139  * Prints this <code>KeySelectorException</code>, its backtrace and
140  * the cause's backtrace to the specified print writer.
141  *
142  * @param s <code>PrintWriter</code> to use for output
143  */
144 public void printStackTrace(PrintWriter s) {
145     super.printStackTrace(s);
146     //XXX print backtrace of cause
147 }
148 }

```

## 8.5 KeySelectorResult.java

Secuencia 95: javax/xml/crypto/KeySelectorResult.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.

```



```
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26  * $Id: KeySelectorResult.java,v 1.3 2005/05/10 15:47:42 mullan Exp $
27  */
28 package javax.xml.crypto;
29
30 import java.security.Key;
31
32 /**
33  * The result returned by the {@link KeySelector#select KeySelector.select}
34  * method.
35  * <p>
36  * At a minimum, a KeySelectorResult contains the Key
37  * selected by the KeySelector. Implementations of this interface
38  * may add methods to return implementation or algorithm specific information,
39  * such as a chain of certificates or debugging information.
40  *
41  * @author Sean Mullan
42  * @author JSR 105 Expert Group
43  * @since 1.6
44  * @see KeySelector
45  */
46 public interface KeySelectorResult {
47
48     /**
49      * Returns the selected key.
50      *
51      * @return the selected key, or null if none can be found
52      */
53     Key getKey();
54 }
```

## 8.6 MarshalException.java

Secuencia 96: javax/xml/crypto/MarshalException.java

```
1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: MarshalException.java,v 1.5 2005/05/10 15:47:42 mullan Exp $
27 */
28 package javax.xml.crypto;
29
30 import java.io.PrintStream;
31 import java.io.PrintWriter;
32 import javax.xml.crypto.dsig.Manifest;
33 import javax.xml.crypto.dsig.XMLSignature;
34 import javax.xml.crypto.dsig.XMLSignatureFactory;
35 import javax.xml.crypto.dsig.keyinfo.KeyInfo;
36 import javax.xml.crypto.dsig.keyinfo.KeyInfoFactory;
37
38 /**
39  * Indicates an exceptional condition that occurred during the XML
40  * marshalling or unmarshalling process.
41  *
42  * <p>A <code>MarshalException</code> can contain a cause: another
43  * throwable that caused this <code>MarshalException</code> to get thrown.
44  *
45  * @author Sean Mullan
46  * @author JSR 105 Expert Group
47  * @since 1.6
48  * @see XMLSignature#sign(XMLSignContext)
49  * @see XMLSignatureFactory#unmarshalXMLSignature(XMLValidateContext)
50  */
```

```

51 public class MarshalException extends Exception {
52
53     private static final long serialVersionUID = -863185580332643547L;
54
55     /**
56      * The throwable that caused this exception to get thrown, or null if this
57      * exception was not caused by another throwable or if the causative
58      * throwable is unknown.
59      *
60      * @serial
61      */
62     private Throwable cause;
63
64     /**
65      * Constructs a new MarshalException with
66      * null as its detail message.
67      */
68     public MarshalException() {
69         super();
70     }
71
72     /**
73      * Constructs a new MarshalException with the specified
74      * detail message.
75      *
76      * @param message the detail message
77      */
78     public MarshalException(String message) {
79         super(message);
80     }
81
82     /**
83      * Constructs a new MarshalException with the
84      * specified detail message and cause.
85      * 

Note that the detail message associated with
86      * cause is not automatically incorporated in
87      * this exception's detail message.
88      *
89      * @param message the detail message
90      * @param cause the cause (A null value is permitted, and
91      *             indicates that the cause is nonexistent or unknown.)
92      */
93     public MarshalException(String message, Throwable cause) {
94         super(message);
95         this.cause = cause;
96     }
97
98     /**
99      * Constructs a new MarshalException with the specified cause
100      * and a detail message of (cause==null ? null : cause.toString())
101      * (which typically contains the class and detail message of
102      * cause).
103      *


```

```

104     * @param cause the cause (A <tt>null</tt> value is permitted, and
105     *     indicates that the cause is nonexistent or unknown.)
106     */
107     public MarshalException(Throwable cause) {
108         super(cause==null ? null : cause.toString());
109         this.cause = cause;
110     }
111
112     /**
113     * Returns the cause of this <code>MarshalException</code> or
114     * <code>null</code> if the cause is nonexistent or unknown. (The
115     * cause is the throwable that caused this
116     * <code>MarshalException</code> to get thrown.)
117     *
118     * @return the cause of this <code>MarshalException</code> or
119     *     <code>null</code> if the cause is nonexistent or unknown.
120     */
121     public Throwable getCause() {
122         return cause;
123     }
124
125     /**
126     * Prints this <code>MarshalException</code>, its backtrace and
127     * the cause's backtrace to the standard error stream.
128     */
129     public void printStackTrace() {
130         super.printStackTrace();
131         //XXX print backtrace of cause
132     }
133
134     /**
135     * Prints this <code>MarshalException</code>, its backtrace and
136     * the cause's backtrace to the specified print stream.
137     *
138     * @param s <code>PrintStream</code> to use for output
139     */
140     public void printStackTrace(PrintStream s) {
141         super.printStackTrace(s);
142         //XXX print backtrace of cause
143     }
144
145     /**
146     * Prints this <code>MarshalException</code>, its backtrace and
147     * the cause's backtrace to the specified print writer.
148     *
149     * @param s <code>PrintWriter</code> to use for output
150     */
151     public void printStackTrace(PrintWriter s) {
152         super.printStackTrace(s);
153         //XXX print backtrace of cause
154     }
155 }

```

## 8.7 NoSuchMechanismException.java

Secuencia 97: javax/xml/crypto/NoSuchMechanismException.java

```
1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: NoSuchMechanismException.java,v 1.4 2005/05/10 15:47:42 mullan Exp $
27 */
28 package javax.xml.crypto;
29
30 import java.io.PrintStream;
31 import java.io.PrintWriter;
32 import javax.xml.crypto.dsig.Manifest;
33 import javax.xml.crypto.dsig.XMLSignature;
34 import javax.xml.crypto.dsig.XMLSignatureFactory;
35 import javax.xml.crypto.dsig.keyinfo.KeyInfo;
36 import javax.xml.crypto.dsig.keyinfo.KeyInfoFactory;
37
38 /**
39 * This exception is thrown when a particular XML mechanism is requested but
40 * is not available in the environment.
41 *
42 * <p>A <code>NoSuchMechanismException</code> can contain a cause: another
43 * throwable that caused this <code>NoSuchMechanismException</code> to get
44 * thrown.
45 *
46 * @author Sean Mullan
47 * @author JSR 105 Expert Group
48 * @since 1.6
49 * @see XMLSignatureFactory#getInstance XMLSignatureFactory.getInstance
50 * @see KeyInfoFactory#getInstance KeyInfoFactory.getInstance
```

```
51  */
52  public class NoSuchMechanismException extends RuntimeException {
53
54      private static final long serialVersionUID = 4189669069570660166L;
55
56      /**
57       * The throwable that caused this exception to get thrown, or null if this
58       * exception was not caused by another throwable or if the causative
59       * throwable is unknown.
60       *
61       * @serial
62       */
63      private Throwable cause;
64
65      /**
66       * Constructs a new NoSuchMechanismException with
67       * null as its detail message.
68       */
69      public NoSuchMechanismException() {
70          super();
71      }
72
73      /**
74       * Constructs a new NoSuchMechanismException with the
75       * specified detail message.
76       *
77       * @param message the detail message
78       */
79      public NoSuchMechanismException(String message) {
80          super(message);
81      }
82
83      /**
84       * Constructs a new NoSuchMechanismException with the
85       * specified detail message and cause.
86       * 

Note that the detail message associated with
87       * cause is not automatically incorporated in
88       * this exception's detail message.
89       *
90       * @param message the detail message
91       * @param cause the cause (A null value is permitted, and
92       *             indicates that the cause is nonexistent or unknown.)
93       */
94      public NoSuchMechanismException(String message, Throwable cause) {
95          super(message);
96          this.cause = cause;
97      }
98
99      /**
100       * Constructs a new NoSuchMechanismException with the
101       * specified cause and a detail message of
102       * (cause==null ? null : cause.toString()) (which typically
103       * contains the class and detail message of cause).


```

```
104     *
105     * @param cause the cause (A <tt>null</tt> value is permitted, and
106     *       indicates that the cause is nonexistent or unknown.)
107     */
108     public NoSuchMechanismException(Throwable cause) {
109         super(cause==null ? null : cause.toString());
110         this.cause = cause;
111     }
112
113     /**
114     * Returns the cause of this <code>NoSuchMechanismException</code> or
115     * <code>null</code> if the cause is nonexistent or unknown. (The
116     * cause is the throwable that caused this
117     * <code>NoSuchMechanismException</code> to get thrown.)
118     *
119     * @return the cause of this <code>NoSuchMechanismException</code> or
120     *         <code>null</code> if the cause is nonexistent or unknown.
121     */
122     public Throwable getCause() {
123         return cause;
124     }
125
126     /**
127     * Prints this <code>NoSuchMechanismException</code>, its backtrace and
128     * the cause's backtrace to the standard error stream.
129     */
130     public void printStackTrace() {
131         super.printStackTrace();
132         //XXX print backtrace of cause
133     }
134
135     /**
136     * Prints this <code>NoSuchMechanismException</code>, its backtrace and
137     * the cause's backtrace to the specified print stream.
138     *
139     * @param s <code>PrintStream</code> to use for output
140     */
141     public void printStackTrace(PrintStream s) {
142         super.printStackTrace(s);
143         //XXX print backtrace of cause
144     }
145
146     /**
147     * Prints this <code>NoSuchMechanismException</code>, its backtrace and
148     * the cause's backtrace to the specified print writer.
149     *
150     * @param s <code>PrintWriter</code> to use for output
151     */
152     public void printStackTrace(PrintWriter s) {
153         super.printStackTrace(s);
154         //XXX print backtrace of cause
155     }
156 }
```

## 8.8 NodeSetData.java

Secuencia 98: javax/xml/crypto/NodeSetData.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: NodeSetData.java,v 1.5 2005/05/10 15:47:42 mullan Exp $
27 */
28 package javax.xml.crypto;
29
30 import java.util.Iterator;
31
32 /**
33 * An abstract representation of a Data type containing a
34 * node-set. The type (class) and ordering of the nodes contained in the set
35 * are not defined by this class; instead that behavior should be
36 * defined by NodeSetData subclasses.
37 *
38 * @author Sean Mullan
39 * @author JSR 105 Expert Group
40 * @since 1.6
41 */
42 public interface NodeSetData extends Data {
43
44     /**
45      * Returns a read-only iterator over the nodes contained in this
46      * NodeSetData in
47      * http://www.w3.org/TR/1999/REC-xpath-19991116#dt-document-order
48      * document order. Attempts to modify the returned iterator
49      * via the remove method throw
50      * UnsupportedOperationException.
```



```

51      *
52      * @return an Iterator over the nodes in this
53      *      NodeSetData in document order
54      */
55      @SuppressWarnings("rawtypes")
56      Iterator iterator();
57  }

```

## 8.9 OctetStreamData.java

Secuencia 99: javax/xml/crypto/OctetStreamData.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: OctetStreamData.java,v 1.3 2005/05/10 15:47:42 mullan Exp $
27 */
28 package javax.xml.crypto;
29
30 import java.io.InputStream;
31
32 /**
33  * A representation of a Data type containing an octet stream.
34  *
35  * @since 1.6
36  */
37 public class OctetStreamData implements Data {
38
39     private InputStream octetStream;
40     private String uri;
41     private String mimeType;
42

```

```
43  /**
44   * Creates a new <code>OctetStreamData</code>.
45   *
46   * @param octetStream the input stream containing the octets
47   * @throws NullPointerException if <code>octetStream</code> is
48   *   <code>null</code>
49   */
50  public OctetStreamData(InputStream octetStream) {
51      if (octetStream == null) {
52          throw new NullPointerException("octetStream is null");
53      }
54      this.octetStream = octetStream;
55  }
56
57  /**
58   * Creates a new <code>OctetStreamData</code>.
59   *
60   * @param octetStream the input stream containing the octets
61   * @param uri the URI String identifying the data object (may be
62   *   <code>null</code>)
63   * @param mimeType the MIME type associated with the data object (may be
64   *   <code>null</code>)
65   * @throws NullPointerException if <code>octetStream</code> is
66   *   <code>null</code>
67   */
68  public OctetStreamData(InputStream octetStream, String uri,
69      String mimeType) {
70      if (octetStream == null) {
71          throw new NullPointerException("octetStream is null");
72      }
73      this.octetStream = octetStream;
74      this.uri = uri;
75      this.mimeType = mimeType;
76  }
77
78  /**
79   * Returns the input stream of this <code>OctetStreamData</code>.
80   *
81   * @return the input stream of this <code>OctetStreamData</code>.
82   */
83  public InputStream getOctetStream() {
84      return octetStream;
85  }
86
87  /**
88   * Returns the URI String identifying the data object represented by this
89   * <code>OctetStreamData</code>.
90   *
91   * @return the URI String or <code>null</code> if not applicable
92   */
93  public String getURI() {
94      return uri;
95  }
```

```

96
97  /**
98   * Returns the MIME type associated with the data object represented by this
99   * <code>OctetStreamData</code>.
100  *
101  * @return the MIME type or <code>null</code> if not applicable
102  */
103  public String getMimeType() {
104      return mimeType;
105  }
106 }

```

## 8.10 URIDereferencer.java

Secuencia 100: javax/xml/crypto/URIDereferencer.java

```

1  /*
2   * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  /*
27   * =====
28   *
29   * (C) Copyright IBM Corp. 2003 All Rights Reserved.
30   *
31   * =====
32  */
33  /*
34   * $Id: URIDereferencer.java,v 1.5 2005/05/10 15:47:42 mullan Exp $
35   */
36  package javax.xml.crypto;
37
38  /**

```

```

39  * A dereferencer of {@link URIReference}s.
40  * <p>
41  * The result of dereferencing a <code>URIReference</code> is either an
42  * instance of {@link OctetStreamData} or {@link NodeSetData}. Unless the
43  * <code>URIReference</code> is a <i>same-document reference</i> as defined
44  * in section 4.2 of the W3C Recommendation for XML-Signature Syntax and
45  * Processing, the result of dereferencing the <code>URIReference</code>
46  * MUST be an <code>OctetStreamData</code>.
47  *
48  * @author Sean Mullan
49  * @author Joyce Leung
50  * @author JSR 105 Expert Group
51  * @since 1.6
52  * @see XMLCryptoContext#setURIDereferencer(URIDereferencer)
53  * @see XMLCryptoContext#getURIDereferencer
54  */
55 public interface URIDereferencer {
56
57     /**
58      * Dereferences the specified <code>URIReference</code> and returns the
59      * dereferenced data.
60      *
61      * @param uriReference the <code>URIReference</code>
62      * @param context an <code>XMLCryptoContext</code> that may
63      *     contain additional useful information for dereferencing the URI. This
64      *     implementation should dereference the specified
65      *     <code>URIReference</code> against the context's <code>baseURI</code>
66      *     parameter, if specified.
67      * @return the dereferenced data
68      * @throws NullPointerException if <code>uriReference</code> or
69      *     <code>context</code> are <code>>null</code>
70      * @throws URIReferenceException if an exception occurs while
71      *     dereferencing the specified <code>uriReference</code>
72      */
73     Data dereference(URIReference uriReference, XMLCryptoContext context)
74         throws URIReferenceException;
75 }

```

## 8.11 URIReference.java

Secuencia 101: javax/xml/crypto/URIReference.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```

13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: URIReference.java,v 1.4 2005/05/10 15:47:42 mullan Exp $
27  */
28  package javax.xml.crypto;
29
30  /**
31   * Identifies a data object via a URI-Reference, as specified by
32   * <a href="http://www.ietf.org/rfc/rfc2396.txt">RFC 2396</a>.
33   *
34   * <p>Note that some subclasses may not have a <code>type</code> attribute
35   * and for objects of those types, the {@link #getType} method always returns
36   * <code>null</code>.
37   *
38   * @author Sean Mullan
39   * @author JSR 105 Expert Group
40   * @since 1.6
41   * @see URIDereferencer
42   */
43  public interface URIReference {
44
45      /**
46       * Returns the URI of the referenced data object.
47       *
48       * @return the URI of the data object in RFC 2396 format (may be
49       *         <code>null</code> if not specified)
50       */
51      String getURI();
52
53      /**
54       * Returns the type of data referenced by this URI.
55       *
56       * @return the type (a URI) of the data object (may be <code>null</code>
57       *         if not specified)
58       */
59      String getType();
60  }

```

## 8.12 URIReferenceException.java

Secuencia 102: javax/xml/crypto/URIReferenceException.java

```

1  /*

```

```
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26  * $Id: URIReferenceException.java,v 1.4 2005/05/10 15:47:42 mullan Exp $
27  */
28 package javax.xml.crypto;
29
30 import java.io.PrintStream;
31 import java.io.PrintWriter;
32 import javax.xml.crypto.dsig.keyinfo.RetrievalMethod;
33
34 /**
35  * Indicates an exceptional condition thrown while dereferencing a
36  * {@link URIReference}.
37  *
38  * <p>A <code>URIReferenceException</code> can contain a cause: another
39  * throwable that caused this <code>URIReferenceException</code> to get thrown.
40  *
41  * @author Sean Mullan
42  * @author JSR 105 Expert Group
43  * @since 1.6
44  * @see URIDereferencer#dereference(URIReference, XMLCryptoContext)
45  * @see RetrievalMethod#dereference(XMLCryptoContext)
46  */
47 public class URIReferenceException extends Exception {
48
49     private static final long serialVersionUID = 7173469703932561419L;
50
51     /**
52      * The throwable that caused this exception to get thrown, or null if this
53      * exception was not caused by another throwable or if the causative
54      * throwable is unknown.
```

```
55     *
56     * @serial
57     */
58     private Throwable cause;
59
60     private URIReference uriReference;
61
62     /**
63      * Constructs a new URISyntaxException with
64      * null as its detail message.
65      */
66     public URISyntaxException() {
67         super();
68     }
69
70     /**
71      * Constructs a new URISyntaxException with the specified
72      * detail message.
73      *
74      * @param message the detail message
75      */
76     public URISyntaxException(String message) {
77         super(message);
78     }
79
80     /**
81      * Constructs a new URISyntaxException with the
82      * specified detail message and cause.
83      * 

Note that the detail message associated with
84      * cause is not automatically incorporated in
85      * this exception's detail message.
86      *
87      * @param message the detail message
88      * @param cause the cause (A null value is permitted, and
89      *             indicates that the cause is nonexistent or unknown.)
90      */
91     public URISyntaxException(String message, Throwable cause) {
92         super(message);
93         this.cause = cause;
94     }
95
96     /**
97      * Constructs a new URISyntaxException with the
98      * specified detail message, cause and URIReference.
99      * 

Note that the detail message associated with
100      * cause is not automatically incorporated in
101      * this exception's detail message.
102      *
103      * @param message the detail message
104      * @param cause the cause (A null value is permitted, and
105      *             indicates that the cause is nonexistent or unknown.)
106      * @param uriReference the URIReference that was being
107      *                    dereferenced when the error was encountered


```

```

108     * @throws NullPointerException if <code>uriReference</code> is
109     *     <code>null</code>
110     */
111     public URIReferenceException(String message, Throwable cause,
112         URIReference uriReference) {
113         this(message, cause);
114         if (uriReference == null) {
115             throw new NullPointerException("uriReference cannot be null");
116         }
117         this.uriReference = uriReference;
118     }
119
120     /**
121     * Constructs a new <code>URIReferenceException</code> with the specified
122     * cause and a detail message of <code>(cause==null ? null :
123     * cause.toString())</code> (which typically contains the class and detail
124     * message of <code>cause</code>).
125     *
126     * @param cause the cause (A <tt>null</tt> value is permitted, and
127     *     indicates that the cause is nonexistent or unknown.)
128     */
129     public URIReferenceException(Throwable cause) {
130         super(cause==null ? null : cause.toString());
131         this.cause = cause;
132     }
133
134     /**
135     * Returns the <code>URIReference</code> that was being dereferenced
136     * when the exception was thrown.
137     *
138     * @return the <code>URIReference</code> that was being dereferenced
139     * when the exception was thrown, or <code>null</code> if not specified
140     */
141     public URIReference getURIReference() {
142         return uriReference;
143     }
144
145     /**
146     * Returns the cause of this <code>URIReferenceException</code> or
147     * <code>null</code> if the cause is nonexistent or unknown. (The
148     * cause is the throwable that caused this
149     * <code>URIReferenceException</code> to get thrown.)
150     *
151     * @return the cause of this <code>URIReferenceException</code> or
152     *     <code>null</code> if the cause is nonexistent or unknown.
153     */
154     public Throwable getCause() {
155         return cause;
156     }
157
158     /**
159     * Prints this <code>URIReferenceException</code>, its backtrace and
160     * the cause's backtrace to the standard error stream.

```



```

161     */
162     public void printStackTrace() {
163         super.printStackTrace();
164         //XXX print backtrace of cause
165     }
166
167     /**
168     * Prints this <code>URIReferenceException</code>, its backtrace and
169     * the cause's backtrace to the specified print stream.
170     *
171     * @param s <code>PrintStream</code> to use for output
172     */
173     public void printStackTrace(PrintStream s) {
174         super.printStackTrace(s);
175         //XXX print backtrace of cause
176     }
177
178     /**
179     * Prints this <code>URIReferenceException</code>, its backtrace and
180     * the cause's backtrace to the specified print writer.
181     *
182     * @param s <code>PrintWriter</code> to use for output
183     */
184     public void printStackTrace(PrintWriter s) {
185         super.printStackTrace(s);
186         //XXX print backtrace of cause
187     }
188 }

```

## 8.13 XMLCryptoContext.java

Secuencia 103: javax/xml/crypto/XMLCryptoContext.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```
22  *
23  *
24  */
25  /*
26  * $Id: XMLCryptoContext.java,v 1.6 2005/05/10 15:47:42 mullan Exp $
27  */
28  package javax.xml.crypto;
29
30  /**
31   * Contains common context information for XML cryptographic operations.
32   *
33   * <p>This interface contains methods for setting and retrieving properties
34   * that affect the processing of XML signatures or XML encrypted structures.
35   *
36   * <p>Note that <code>XMLCryptoContext</code> instances can contain information
37   * and state specific to the XML cryptographic structure it is used with.
38   * The results are unpredictable if an <code>XMLCryptoContext</code> is
39   * used with multiple structures (for example, you should not use the same
40   * {@link javax.xml.crypto.dsig.XMLValidateContext} instance to validate two
41   * different {@link javax.xml.crypto.dsig.XMLSignature} objects).
42   *
43   * @author Sean Mullan
44   * @author JSR 105 Expert Group
45   * @since 1.6
46   */
47  public interface XMLCryptoContext {
48
49      /**
50       * Returns the base URI.
51       *
52       * @return the base URI, or <code>null</code> if not specified
53       * @see #setBaseURI(String)
54       */
55      String getBaseURI();
56
57      /**
58       * Sets the base URI.
59       *
60       * @param baseURI the base URI, or <code>null</code> to remove current
61       *        value
62       * @throws IllegalArgumentException if <code>baseURI</code> is not RFC
63       *        2396 compliant
64       * @see #getBaseURI
65       */
66      void setBaseURI(String baseURI);
67
68      /**
69       * Returns the key selector for finding a key.
70       *
71       * @return the key selector, or <code>null</code> if not specified
72       * @see #setKeySelector(KeySelector)
73       */
74      KeySelector getKeySelector();
```

```

75
76 /**
77  * Sets the key selector for finding a key.
78  *
79  * @param ks the key selector, or null to remove the current
80  *   setting
81  * @see #getKeySelector
82  */
83 void setKeySelector(KeySelector ks);
84
85 /**
86  * Returns a URIDereferencer that is used to dereference
87  * {@link URIReference}s.
88  *
89  * @return the URIDereferencer, or null if not
90  *   specified
91  * @see #setURIDereferencer(URIDereferencer)
92  */
93 URIDereferencer getURIDereferencer();
94
95 /**
96  * Sets a URIDereferencer that is used to dereference
97  * {@link URIReference}s. The specified URIDereferencer
98  * is used in place of an implementation's default
99  * URIDereferencer.
100  *
101  * @param dereferencer the URIDereferencer, or
102  *   null to remove any current setting
103  * @see #getURIDereferencer
104  */
105 void setURIDereferencer(URIDereferencer dereferencer);
106
107 /**
108  * Returns the namespace prefix that the specified namespace URI is
109  * associated with. Returns the specified default prefix if the specified
110  * namespace URI has not been bound to a prefix. To bind a namespace URI
111  * to a prefix, call the {@link #putNamespacePrefix putNamespacePrefix}
112  * method.
113  *
114  * @param namespaceURI a namespace URI
115  * @param defaultPrefix the prefix to be returned in the event that the
116  *   the specified namespace URI has not been bound to a prefix.
117  * @return the prefix that is associated with the specified namespace URI,
118  *   or defaultPrefix if the URI is not registered. If
119  *   the namespace URI is registered but has no prefix, an empty string
120  *   ("") is returned.
121  * @throws NullPointerException if namespaceURI is
122  *   null
123  * @see #putNamespacePrefix(String, String)
124  */
125 String getNamespacePrefix(String namespaceURI, String defaultPrefix);
126
127 /**

```

```

128     * Maps the specified namespace URI to the specified prefix. If there is
129     * already a prefix associated with the specified namespace URI, the old
130     * prefix is replaced by the specified prefix.
131     *
132     * @param namespaceURI a namespace URI
133     * @param prefix a namespace prefix (or null to remove any
134     *     existing mapping). Specifying the empty string ("")
135     *     binds no prefix to the namespace URI.
136     * @return the previous prefix associated with the specified namespace
137     *     URI, or null if there was none
138     * @throws NullPointerException if namespaceURI is
139     *     null
140     * @see #getNamespacePrefix(String, String)
141     */
142     String putNamespacePrefix(String namespaceURI, String prefix);
143
144     /**
145     * Returns the default namespace prefix. The default namespace prefix
146     * is the prefix for all namespace URIs not explicitly set by the
147     * {@link #putNamespacePrefix putNamespacePrefix} method.
148     *
149     * @return the default namespace prefix, or null if none has
150     *     been set.
151     * @see #setDefaultNamespacePrefix(String)
152     */
153     String getDefaultNamespacePrefix();
154
155     /**
156     * Sets the default namespace prefix. This sets the namespace prefix for
157     * all namespace URIs not explicitly set by the {@link #putNamespacePrefix
158     * putNamespacePrefix} method.
159     *
160     * @param defaultPrefix the default namespace prefix, or null
161     *     to remove the current setting. Specify the empty string
162     *     ("") to bind no prefix.
163     * @see #getDefaultNamespacePrefix
164     */
165     void setDefaultNamespacePrefix(String defaultPrefix);
166
167     /**
168     * Sets the specified property.
169     *
170     * @param name the name of the property
171     * @param value the value of the property to be set
172     * @return the previous value of the specified property, or
173     *     null if it did not have a value
174     * @throws NullPointerException if name is null
175     * @see #getProperty(String)
176     */
177     Object setProperty(String name, Object value);
178
179     /**
180     * Returns the value of the specified property.

```

```

181      *
182      * @param name the name of the property
183      * @return the current value of the specified property, or
184      *         <code>null</code> if it does not have a value
185      * @throws NullPointerException if <code>name</code> is <code>null</code>
186      * @see #setProperty(String, Object)
187      */
188      Object getProperty(String name);
189
190      /**
191       * Returns the value to which this context maps the specified key.
192       *
193       * <p>More formally, if this context contains a mapping from a key
194       * <code>k</code> to a value <code>v</code> such that
195       * <code>(key==null ? k==null : key.equals(k))</code>, then this method
196       * returns <code>v</code>; otherwise it returns <code>null</code>. (There
197       * can be at most one such mapping.)
198       *
199       * <p>This method is useful for retrieving arbitrary information that is
200       * specific to the cryptographic operation that this context is used for.
201       *
202       * @param key the key whose associated value is to be returned
203       * @return the value to which this context maps the specified key, or
204       *         <code>null</code> if there is no mapping for the key
205       * @see #put(Object, Object)
206       */
207      Object get(Object key);
208
209      /**
210       * Associates the specified value with the specified key in this context.
211       * If the context previously contained a mapping for this key, the old
212       * value is replaced by the specified value.
213       *
214       * <p>This method is useful for storing arbitrary information that is
215       * specific to the cryptographic operation that this context is used for.
216       *
217       * @param key key with which the specified value is to be associated with
218       * @param value value to be associated with the specified key
219       * @return the previous value associated with the key, or <code>null</code>
220       *         if there was no mapping for the key
221       * @throws IllegalArgumentException if some aspect of this key or value
222       *         prevents it from being stored in this context
223       * @see #get(Object)
224       */
225      Object put(Object key, Object value);
226  }

```

## 8.14 XMLStructure.java

Secuencia 104: javax/xml/crypto/XMLStructure.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

```

```

4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26  * $Id: XMLStructure.java,v 1.3 2005/05/10 15:47:42 mullan Exp $
27  */
28 package javax.xml.crypto;
29
30 /**
31  * A representation of an XML structure from any namespace. The purpose of
32  * this interface is to group (and provide type safety for) all
33  * representations of XML structures.
34  *
35  * @author Sean Mullan
36  * @author JSR 105 Expert Group
37  * @since 1.6
38  */
39 public interface XMLStructure {
40
41     /**
42      * Indicates whether a specified feature is supported.
43      *
44      * @param feature the feature name (as an absolute URI)
45      * @return <code>true</code> if the specified feature is supported,
46      *         <code>false</code> otherwise
47      * @throws NullPointerException if <code>feature</code> is <code>null</code>
48      */
49     boolean isFeatureSupported(String feature);
50 }

```

## 9 Xml Crypto Dom (javax.xml.crypto.dom package)

### 9.1 DOMCryptoContext.java

Secuencia 105: javax/xml/crypto/dom/DOMCryptoContext.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: DOMCryptoContext.java,v 1.3 2005/05/09 18:33:26 mullan Exp $
27 */
28 package javax.xml.crypto.dom;
29
30 import javax.xml.crypto.KeySelector;
31 import javax.xml.crypto.URIDereferencer;
32 import javax.xml.crypto.XMLCryptoContext;
33 import java.util.Collections;
34 import java.util.HashMap;
35 import java.util.Iterator;
36 import org.w3c.dom.Element;
37
38 /**
39 * This class provides a DOM-specific implementation of the
40 * {@link XMLCryptoContext} interface. It also includes additional
41 * methods that are specific to a DOM-based implementation for registering
42 * and retrieving elements that contain attributes of type ID.
43 *
44 * @author Sean Mullan
45 * @author JSR 105 Expert Group
46 * @since 1.6
47 */
48 public class DOMCryptoContext implements XMLCryptoContext {
49
50     private HashMap<String,String> nsMap = new HashMap<>();
51     private HashMap<String,Element> idMap = new HashMap<>();
52     private HashMap<Object,Object> objMap = new HashMap<>();
53     private String baseURI;
```

```
54 private KeySelector ks;
55 private URIDereferencer dereferencer;
56 private HashMap<String, Object> propMap = new HashMap<>();
57 private String defaultPrefix;
58
59 /**
60  * Default constructor. (For invocation by subclass constructors).
61  */
62 protected DOMCryptoContext() {}
63
64 /**
65  * This implementation uses an internal {@link HashMap} to get the prefix
66  * that the specified URI maps to. It returns the <code>defaultPrefix</code>
67  * if it maps to <code>null</code>.
68  *
69  * @throws NullPointerException {@inheritDoc}
70  */
71 public String getNamespacePrefix(String namespaceURI,
72     String defaultPrefix) {
73     if (namespaceURI == null) {
74         throw new NullPointerException("namespaceURI cannot be null");
75     }
76     String prefix = nsMap.get(namespaceURI);
77     return (prefix != null ? prefix : defaultPrefix);
78 }
79
80 /**
81  * This implementation uses an internal {@link HashMap} to map the URI
82  * to the specified prefix.
83  *
84  * @throws NullPointerException {@inheritDoc}
85  */
86 public String putNamespacePrefix(String namespaceURI, String prefix) {
87     if (namespaceURI == null) {
88         throw new NullPointerException("namespaceURI is null");
89     }
90     return nsMap.put(namespaceURI, prefix);
91 }
92
93 public String getDefaultNamespacePrefix() {
94     return defaultPrefix;
95 }
96
97 public void setDefaultNamespacePrefix(String defaultPrefix) {
98     this.defaultPrefix = defaultPrefix;
99 }
100
101 public String getBaseURI() {
102     return baseURI;
103 }
104
105 /**
106  * @throws IllegalArgumentException {@inheritDoc}
```



```
107     */
108     public void setBaseURI(String baseURI) {
109         if (baseURI != null) {
110             java.net.URI.create(baseURI);
111         }
112         this.baseURI = baseURI;
113     }
114
115     public URIDereferencer getURIDereferencer() {
116         return dereferencer;
117     }
118
119     public void setURIDereferencer(URIDereferencer dereferencer) {
120         this.dereferencer = dereferencer;
121     }
122
123     /**
124      * This implementation uses an internal {@link HashMap} to get the object
125      * that the specified name maps to.
126      *
127      * @throws NullPointerException {@inheritDoc}
128      */
129     public Object getProperty(String name) {
130         if (name == null) {
131             throw new NullPointerException("name is null");
132         }
133         return propMap.get(name);
134     }
135
136     /**
137      * This implementation uses an internal {@link HashMap} to map the name
138      * to the specified object.
139      *
140      * @throws NullPointerException {@inheritDoc}
141      */
142     public Object setProperty(String name, Object value) {
143         if (name == null) {
144             throw new NullPointerException("name is null");
145         }
146         return propMap.put(name, value);
147     }
148
149     public KeySelector getKeySelector() {
150         return ks;
151     }
152
153     public void setKeySelector(KeySelector ks) {
154         this.ks = ks;
155     }
156
157     /**
158      * Returns the Element with the specified ID attribute value.
159      *
```

```

160 * <p>This implementation uses an internal {@link HashMap} to get the
161 * element that the specified attribute value maps to.
162 *
163 * @param idValue the value of the ID
164 * @return the <code>Element</code> with the specified ID attribute value,
165 *      or <code>null</code> if none.
166 * @throws NullPointerException if <code>idValue</code> is <code>null</code>
167 * @see #setIdAttributeNS
168 */
169 public Element getElementById(String idValue) {
170     if (idValue == null) {
171         throw new NullPointerException("idValue is null");
172     }
173     return idMap.get(idValue);
174 }
175
176 /**
177 * Registers the element's attribute specified by the namespace URI and
178 * local name to be of type ID. The attribute must have a non-empty value.
179 *
180 * <p>This implementation uses an internal {@link HashMap} to map the
181 * attribute's value to the specified element.
182 *
183 * @param element the element
184 * @param namespaceURI the namespace URI of the attribute (specify
185 *      <code>null</code> if not applicable)
186 * @param localName the local name of the attribute
187 * @throws IllegalArgumentException if <code>localName</code> is not an
188 *      attribute of the specified element or it does not contain a specific
189 *      value
190 * @throws NullPointerException if <code>element</code> or
191 *      <code>localName</code> is <code>null</code>
192 * @see #getElementById
193 */
194 public void setIdAttributeNS(Element element, String namespaceURI,
195     String localName) {
196     if (element == null) {
197         throw new NullPointerException("element is null");
198     }
199     if (localName == null) {
200         throw new NullPointerException("localName is null");
201     }
202     String idValue = element.getAttributeNS(namespaceURI, localName);
203     if (idValue == null || idValue.length() == 0) {
204         throw new IllegalArgumentException(localName + " is not an " +
205             "attribute");
206     }
207     idMap.put(idValue, element);
208 }
209
210 /**
211 * Returns a read-only iterator over the set of Id/Element mappings of
212 * this <code>DOMCryptoContext</code>. Attempts to modify the set via the

```

```

213 * {@link Iterator#remove} method throw an
214 * <code>UnsupportedOperationException</code>. The mappings are returned
215 * in no particular order. Each element in the iteration is represented as a
216 * {@link java.util.Map.Entry}. If the <code>DOMCryptoContext</code> is
217 * modified while an iteration is in progress, the results of the
218 * iteration are undefined.
219 *
220 * @return a read-only iterator over the set of mappings
221 */
222 @SuppressWarnings("rawtypes")
223 public Iterator iterator() {
224     return Collections.unmodifiableMap(idMap).entrySet().iterator();
225 }
226
227 /**
228 * This implementation uses an internal {@link HashMap} to get the object
229 * that the specified key maps to.
230 */
231 public Object get(Object key) {
232     return objMap.get(key);
233 }
234
235 /**
236 * This implementation uses an internal {@link HashMap} to map the key
237 * to the specified object.
238 *
239 * @throws IllegalArgumentException {@@inheritDoc}
240 */
241 public Object put(Object key, Object value) {
242     return objMap.put(key, value);
243 }
244 }

```

## 9.2 DOMStructure.java

Secuencia 106: javax/xml/crypto/dom/DOMStructure.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```

18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: DOMStructure.java,v 1.6 2005/05/09 18:33:26 mullan Exp $
27  */
28  package javax.xml.crypto.dom;
29
30  import org.w3c.dom.Node;
31  import javax.xml.crypto.XMLStructure;
32  import javax.xml.crypto.dsig.XMLSignature;
33
34  /**
35   * A DOM-specific {@link XMLStructure}. The purpose of this class is to
36   * allow a DOM node to be used to represent extensible content (any elements
37   * or mixed content) in XML Signature structures.
38   *
39   * <p>If a sequence of nodes is needed, the node contained in the
40   * <code>DOMStructure</code> is the first node of the sequence and successive
41   * nodes can be accessed by invoking {@link Node#getNextSibling}.
42   *
43   * <p>If the owner document of the <code>DOMStructure</code> is different than
44   * the target document of an <code>XMLSignature</code>, the
45   * {@link XMLSignature#sign(XMLSignContext)} method imports the node into the
46   * target document before generating the signature.
47   *
48   * @author Sean Mullan
49   * @author JSR 105 Expert Group
50   * @since 1.6
51   */
52  public class DOMStructure implements XMLStructure {
53
54      private final Node node;
55
56      /**
57       * Creates a <code>DOMStructure</code> containing the specified node.
58       *
59       * @param node the node
60       * @throws NullPointerException if <code>node</code> is <code>null</code>
61       */
62      public DOMStructure(Node node) {
63          if (node == null) {
64              throw new NullPointerException("node cannot be null");
65          }
66          this.node = node;
67      }
68
69      /**
70       * Returns the node contained in this <code>DOMStructure</code>.

```

```

71      *
72      * @return the node
73      */
74      public Node getNode() {
75          return node;
76      }
77
78      /**
79      * @throws NullPointerException {@inheritDoc}
80      */
81      public boolean isFeatureSupported(String feature) {
82          if (feature == null) {
83              throw new NullPointerException();
84          } else {
85              return false;
86          }
87      }
88  }

```

### 9.3 DOMURIReference.java

Secuencia 107: javax/xml/crypto/dom/DOMURIReference.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: DOMURIReference.java,v 1.5 2005/05/09 18:33:26 mullan Exp $
27 */
28 package javax.xml.crypto.dom;
29
30 import javax.xml.crypto.URIReference;
31 import org.w3c.dom.Node;

```

```

32
33 /**
34  * A DOM-specific {@link URIReference}. The purpose of this class is to
35  * provide additional context necessary for resolving XPointer URIs or
36  * same-document references.
37  *
38  * @author Sean Mullan
39  * @author JSR 105 Expert Group
40  * @since 1.6
41  */
42 public interface DOMURIReference extends URIReference {
43
44     /**
45      * Returns the here node.
46      *
47      * @return the attribute or processing instruction node or the
48      *         parent element of the text node that directly contains the URI
49      */
50     Node getHere();
51 }

```

## 10 Xml Crypto Dsig (javax.xml.crypto.dsig package)

### 10.1 CanonicalizationMethod.java

Secuencia 108: javax/xml/crypto/dsig/CanonicalizationMethod.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: CanonicalizationMethod.java,v 1.6 2005/05/10 16:03:45 mullan Exp $
27 */

```

```

28 package javax.xml.crypto.dsig;
29
30 import java.security.spec.AlgorithmParameterSpec;
31 import javax.xml.crypto.dsig.spec.C14NMethodParameterSpec;
32
33 /**
34  * A representation of the XML CanonicalizationMethod
35  * element as defined in the
36  * http://www.w3.org/TR/xmlldsig-core/
37  * W3C Recommendation for XML-Signature Syntax and Processing. The XML
38  * Schema Definition is defined as:
39  * 

```

40  * <element name="CanonicalizationMethod" type="ds:CanonicalizationMethodType"/>
41  *   <complexType name="CanonicalizationMethodType" mixed="true">
42  *     <sequence>
43  *       <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
44  *       <!-- (0,unbounded) elements from (1,1) namespace -->
45  *     </sequence>
46  *     <attribute name="Algorithm" type="anyURI" use="required"/>
47  *   </complexType>
48  * </pre>
49  *
50  * A CanonicalizationMethod instance may be created by invoking
51  * the {@link XMLSignatureFactory#newCanonicalizationMethod
52  * newCanonicalizationMethod} method of the {@link XMLSignatureFactory} class.
53  *
54  * @author Sean Mullan
55  * @author JSR 105 Expert Group
56  * @since 1.6
57  * @see XMLSignatureFactory#newCanonicalizationMethod(String, C14NMethodParameterSpec)
58  */
59
60 public interface CanonicalizationMethod extends Transform {
61
62     /**
63      * The http://www.w3.org/TR/2001/REC-xml-c14n-20010315 Canonical
64      * XML (without comments) canonicalization method algorithm URI.
65      */
66     final static String INCLUSIVE =
67         "http://www.w3.org/TR/2001/REC-xml-c14n-20010315";
68
69     /**
70      * The
71      * http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
72      * Canonical XML with comments canonicalization method algorithm URI.
73      */
74     final static String INCLUSIVE_WITH_COMMENTS =
75         "http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments";
76
77     /**
78      * The http://www.w3.org/2001/10/xml-exc-c14n# Exclusive
79      * Canonical XML (without comments) canonicalization method algorithm
80      * URI.

```


```

```

81  */
82  final static String EXCLUSIVE =
83      "http://www.w3.org/2001/10/xml-exc-c14n#";
84
85  /**
86   * The <a href="http://www.w3.org/2001/10/xml-exc-c14n#WithComments">
87   * Exclusive Canonical XML with comments</a> canonicalization method
88   * algorithm URI.
89   */
90  final static String EXCLUSIVE_WITH_COMMENTS =
91      "http://www.w3.org/2001/10/xml-exc-c14n#WithComments";
92
93  /**
94   * Returns the algorithm-specific input parameters associated with this
95   * <code>CanonicalizationMethod</code>.
96   *
97   * <p>The returned parameters can be typecast to a
98   * {@link C14NMethodParameterSpec} object.
99   *
100  * @return the algorithm-specific input parameters (may be
101  *         <code>null</code> if not specified)
102  */
103  AlgorithmParameterSpec getParameterSpec();
104 }

```

## 10.2 DigestMethod.java

Secuencia 109: javax/xml/crypto/dsig/DigestMethod.java

```

1  /*
2   * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*

```



```

26  * $Id: DigestMethod.java,v 1.6 2005/05/10 16:03:46 mullan Exp $
27  */
28  package javax.xml.crypto.dsig;
29
30  import javax.xml.crypto.AlgorithmMethod;
31  import javax.xml.crypto.XMLStructure;
32  import javax.xml.crypto.dsig.spec.DigestMethodParameterSpec;
33  import java.security.spec.AlgorithmParameterSpec;
34
35  /**
36   * A representation of the XML <code>DigestMethod</code> element as
37   * defined in the <a href="http://www.w3.org/TR/xmldsig-core/">
38   * W3C Recommendation for XML-Signature Syntax and Processing</a>.
39   * The XML Schema Definition is defined as:
40   * <p>
41   * <pre>
42   *   <lt;element name="DigestMethod" type="ds:DigestMethodType"/>
43   *   <lt;complexType name="DigestMethodType" mixed="true">
44   *     <lt;sequence>
45   *       <lt;any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
46   *       <lt;!-- (0,unbounded) elements from (1,1) namespace -->
47   *     </sequence>
48   *     <lt;attribute name="Algorithm" type="anyURI" use="required"/>
49   *   </complexType>
50   * </pre>
51   *
52   * A <code>DigestMethod</code> instance may be created by invoking the
53   * {@link XMLSignatureFactory#newDigestMethod newDigestMethod} method
54   * of the {@link XMLSignatureFactory} class.
55   *
56   * @author Sean Mullan
57   * @author JSR 105 Expert Group
58   * @since 1.6
59   * @see XMLSignatureFactory#newDigestMethod(String, DigestMethodParameterSpec)
60   */
61  public interface DigestMethod extends XMLStructure, AlgorithmMethod {
62
63      /**
64       * The <a href="http://www.w3.org/2000/09/xmldsig#sha1">
65       * SHA1</a> digest method algorithm URI.
66       */
67      static final String SHA1 = "http://www.w3.org/2000/09/xmldsig#sha1";
68
69      /**
70       * The <a href="http://www.w3.org/2001/04/xmlenc#sha256">
71       * SHA256</a> digest method algorithm URI.
72       */
73      static final String SHA256 = "http://www.w3.org/2001/04/xmlenc#sha256";
74
75      /**
76       * The <a href="http://www.w3.org/2001/04/xmlenc#sha512">
77       * SHA512</a> digest method algorithm URI.
78       */

```

```

79     static final String SHA512 = "http://www.w3.org/2001/04/xmlenc#sha512";
80
81     /**
82      * The <a href="http://www.w3.org/2001/04/xmlenc#ripemd160">
83      * RIPEMD-160</a> digest method algorithm URI.
84      */
85     static final String RIPEMD160 = "http://www.w3.org/2001/04/xmlenc#ripemd160";
86
87     /**
88      * Returns the algorithm-specific input parameters associated with this
89      * <code>DigestMethod</code>.
90      *
91      * <p>The returned parameters can be typecast to a {@link
92      * DigestMethodParameterSpec} object.
93      *
94      * @return the algorithm-specific parameters (may be <code>null</code> if
95      *         not specified)
96      */
97     AlgorithmParameterSpec getParameterSpec();
98 }

```

### 10.3 Manifest.java

Secuencia 110: javax/xml/crypto/dsig/Manifest.java

```

1  /*
2   * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25 /*
26  * $Id: Manifest.java,v 1.7 2005/05/10 16:03:46 mullan Exp $
27  */
28 package javax.xml.crypto.dsig;
29

```

```

30 import javax.xml.crypto.XMLStructure;
31 import java.util.List;
32
33 /**
34  * A representation of the XML <code>Manifest</code> element as defined in
35  * the <a href="http://www.w3.org/TR/xmlsig-core/">
36  * W3C Recommendation for XML-Signature Syntax and Processing</a>.
37  * The XML Schema Definition is defined as:
38  * <pre>{@code
39  * <element name="Manifest" type="ds:ManifestType"/>
40  *   <complexType name="ManifestType">
41  *     <sequence>
42  *       <element ref="ds:Reference" maxOccurs="unbounded"/>
43  *     </sequence>
44  *     <attribute name="Id" type="ID" use="optional"/>
45  *   </complexType>
46  * }</pre>
47  *
48  * A <code>Manifest</code> instance may be created by invoking
49  * one of the {@link XMLSignatureFactory#newManifest newManifest}
50  * methods of the {@link XMLSignatureFactory} class; for example:
51  *
52  * <pre>
53  * XMLSignatureFactory factory = XMLSignatureFactory.getInstance("DOM");
54  * List references = Collections.singletonList(factory.newReference
55  *     ("#reference-1", DigestMethod.SHA1));
56  * Manifest manifest = factory.newManifest(references, "manifest-1");
57  * </pre>
58  *
59  * @author Sean Mullan
60  * @author JSR 105 Expert Group
61  * @since 1.6
62  * @see XMLSignatureFactory#newManifest(List)
63  * @see XMLSignatureFactory#newManifest(List, String)
64  */
65 public interface Manifest extends XMLStructure {
66
67     /**
68      * URI that identifies the <code>Manifest</code> element (this can be
69      * specified as the value of the <code>type</code> parameter of the
70      * {@link Reference} class to identify the referent's type).
71      */
72     final static String TYPE = "http://www.w3.org/2000/09/xmlsig#Manifest";
73
74     /**
75      * Returns the Id of this <code>Manifest</code>.
76      *
77      * @return the Id of this <code>Manifest</code> (or <code>null</code>
78      *         if not specified)
79      */
80     String getId();
81
82     /**

```

```

83     * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
84     * list} of one or more {@link Reference}s that are contained in this
85     * Manifest.
86     *
87     * @return an unmodifiable list of one or more References
88     */
89     @SuppressWarnings("rawtypes")
90     List getReferences();
91 }

```

## 10.4 Reference.java

Secuencia 111: javax/xml/crypto/dsig/Reference.java

```

1  /*
2   * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25 /*
26  * $Id: Reference.java,v 1.9 2005/05/10 16:03:46 mullan Exp $
27  */
28 package javax.xml.crypto.dsig;
29
30 import javax.xml.crypto.Data;
31 import javax.xml.crypto.URIReference;
32 import javax.xml.crypto.XMLStructure;
33 import java.io.InputStream;
34 import java.util.List;
35
36 /**
37  * A representation of the Reference element as defined in the
38  * http://www.w3.org/TR/xmlsig-core/
39  * W3C Recommendation for XML-Signature Syntax and Processing.
40  * The XML schema is defined as:

```

```

41 * <code><pre>
42 * <lt;element name="Reference" type="ds:ReferenceType"/>
43 * <lt;complexType name="ReferenceType">
44 *   <lt;sequence>
45 *     <lt;element ref="ds:Transforms" minOccurs="0"/>
46 *     <lt;element ref="ds:DigestMethod"/>
47 *     <lt;element ref="ds:DigestValue"/>
48 *   <lt;/sequence>
49 *   <lt;attribute name="Id" type="ID" use="optional"/>
50 *   <lt;attribute name="URI" type="anyURI" use="optional"/>
51 *   <lt;attribute name="Type" type="anyURI" use="optional"/>
52 * <lt;/complexType>
53 *
54 * <lt;element name="DigestValue" type="ds:DigestValueType"/>
55 * <lt;simplerType name="DigestValueType">
56 *   <lt;restriction base="base64Binary"/>
57 * <lt;/simplerType>
58 * </pre></code>
59 *
60 * <p>A <code>Reference</code> instance may be created by invoking one of the
61 * {@link XMLSignatureFactory#newReference newReference} methods of the
62 * {@link XMLSignatureFactory} class; for example:
63 *
64 * <pre>
65 * XMLSignatureFactory factory = XMLSignatureFactory.getInstance("DOM");
66 * Reference ref = factory.newReference
67 *   ("http://www.ietf.org/rfc/rfc3275.txt",
68 *    factory.newDigestMethod(DigestMethod.SHA1, null));
69 * </pre>
70 *
71 * @author Sean Mullan
72 * @author Erwin van der Koogh
73 * @author JSR 105 Expert Group
74 * @since 1.6
75 * @see XMLSignatureFactory#newReference(String, DigestMethod)
76 * @see XMLSignatureFactory#newReference(String, DigestMethod, List, String, String)
77 */
78 public interface Reference extends URIReference, XMLStructure {
79
80   /**
81    * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
82    * list} of {@link Transform}s that are contained in this
83    * <code>Reference</code>.
84    *
85    * @return an unmodifiable list of <code>Transform</code>s
86    *   (may be empty but never <code>null</code>)
87    */
88   @SuppressWarnings("rawtypes")
89   List getTransforms();
90
91   /**
92    * Returns the digest method of this <code>Reference</code>.
93    *

```

```

94     * @return the digest method
95     */
96     DigestMethod getDigestMethod();
97
98     /**
99     * Returns the optional <code>Id</code> attribute of this
100    * <code>Reference</code>, which permits this reference to be
101    * referenced from elsewhere.
102    *
103    * @return the <code>Id</code> attribute (may be <code>null</code> if not
104    *     specified)
105    */
106    String getId();
107
108    /**
109    * Returns the digest value of this <code>Reference</code>.
110    *
111    * @return the raw digest value, or <code>null</code> if this reference has
112    *     not been digested yet. Each invocation of this method returns a new
113    *     clone to protect against subsequent modification.
114    */
115    byte[] getDigestValue();
116
117    /**
118    * Returns the calculated digest value of this <code>Reference</code>
119    * after a validation operation. This method is useful for debugging if
120    * the reference fails to validate.
121    *
122    * @return the calculated digest value, or <code>null</code> if this
123    *     reference has not been validated yet. Each invocation of this method
124    *     returns a new clone to protect against subsequent modification.
125    */
126    byte[] getCalculatedDigestValue();
127
128    /**
129    * Validates this reference. This method verifies the digest of this
130    * reference.
131    *
132    * <p>This method only validates the reference the first time it is
133    * invoked. On subsequent invocations, it returns a cached result.
134    *
135    * @return <code>true</code> if this reference was validated successfully;
136    *     <code>false</code> otherwise
137    * @param validateContext the validating context
138    * @throws NullPointerException if <code>validateContext</code> is
139    *     <code>null</code>
140    * @throws XMLSignatureException if an unexpected exception occurs while
141    *     validating the reference
142    */
143    boolean validate(XMLValidateContext validateContext)
144        throws XMLSignatureException;
145
146    /**

```

```

147     * Returns the dereferenced data, if
148     * <a href="XMLSignContext.html#Supported Properties">reference caching</a>
149     * is enabled. This is the result of dereferencing the URI of this
150     * reference during a validation or generation operation.
151     *
152     * @return the dereferenced data, or <code>null</code> if reference
153     *         caching is not enabled or this reference has not been generated or
154     *         validated
155     */
156     Data getDereferencedData();
157
158     /**
159     * Returns the pre-digested input stream, if
160     * <a href="XMLSignContext.html#Supported Properties">reference caching</a>
161     * is enabled. This is the input to the digest operation during a
162     * validation or signing operation.
163     *
164     * @return an input stream containing the pre-digested input, or
165     *         <code>null</code> if reference caching is not enabled or this
166     *         reference has not been generated or validated
167     */
168     InputStream getDigestInputStream();
169 }

```

## 10.5 SignatureMethod.java

Secuencia 112: javax/xml/crypto/dsig/SignatureMethod.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: SignatureMethod.java,v 1.5 2005/05/10 16:03:46 mullan Exp $

```

```

27  */
28  package javax.xml.crypto.dsig;
29
30  import javax.xml.crypto.AlgorithmMethod;
31  import javax.xml.crypto.XMLStructure;
32  import javax.xml.crypto.dsig.spec.SignatureMethodParameterSpec;
33  import java.security.spec.AlgorithmParameterSpec;
34
35  /**
36   * A representation of the XML <code>SignatureMethod</code> element
37   * as defined in the <a href="http://www.w3.org/TR/xmlldsig-core/">
38   * W3C Recommendation for XML-Signature Syntax and Processing</a>.
39   * The XML Schema Definition is defined as:
40   * <p>
41   * <pre>
42   *   <element name="SignatureMethod" type="ds:SignatureMethodType"/>
43   *   <complexType name="SignatureMethodType" mixed="true">
44   *     <sequence>
45   *       <element name="HMACOutputLength" minOccurs="0" type="ds:HMACOutputLengthType"/>
46   *       <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
47   *       <!-- (0,unbounded) elements from (1,1) namespace -->
48   *     </sequence>
49   *     <attribute name="Algorithm" type="anyURI" use="required"/>
50   *   </complexType>
51   * </pre>
52   *
53   * A <code>SignatureMethod</code> instance may be created by invoking the
54   * {@link XMLSignatureFactory#newSignatureMethod newSignatureMethod} method
55   * of the {@link XMLSignatureFactory} class.
56   *
57   * @author Sean Mullan
58   * @author JSR 105 Expert Group
59   * @since 1.6
60   * @see XMLSignatureFactory#newSignatureMethod(String, SignatureMethodParameterSpec)
61   */
62  public interface SignatureMethod extends XMLStructure, AlgorithmMethod {
63
64      /**
65       * The <a href="http://www.w3.org/2000/09/xmlldsig#dsa-sha1">DSAwithSHA1</a>
66       * (DSS) signature method algorithm URI.
67       */
68      static final String DSA_SHA1 =
69          "http://www.w3.org/2000/09/xmlldsig#dsa-sha1";
70
71      /**
72       * The <a href="http://www.w3.org/2000/09/xmlldsig#rsa-sha1">RSAwithSHA1</a>
73       * (PKCS #1) signature method algorithm URI.
74       */
75      static final String RSA_SHA1 =
76          "http://www.w3.org/2000/09/xmlldsig#rsa-sha1";
77
78      /**
79       * The <a href="http://www.w3.org/2000/09/xmlldsig#hmac-sha1">HMAC-SHA1</a>

```



```

80     * MAC signature method algorithm URI
81     */
82     static final String HMAC_SHA1 =
83         "http://www.w3.org/2000/09/xmlsig#hmac-sha1";
84
85     /**
86     * Returns the algorithm-specific input parameters of this
87     * <code>SignatureMethod</code>.
88     *
89     * <p>The returned parameters can be typecast to a {@link
90     * SignatureMethodParameterSpec} object.
91     *
92     * @return the algorithm-specific input parameters of this
93     *         <code>SignatureMethod</code> (may be <code>null</code> if not
94     *         specified)
95     */
96     AlgorithmParameterSpec getParameterSpec();
97 }

```

## 10.6 SignatureProperties.java

Secuencia 113: javax/xml/crypto/dsig/SignatureProperties.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: SignatureProperties.java,v 1.4 2005/05/10 16:03:46 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import javax.xml.crypto.XMLStructure;
31 import java.util.List;

```

```

32
33 /**
34  * A representation of the XML <code>SignatureProperties</code> element as
35  * defined in the <a href="http://www.w3.org/TR/xmlsig-core/">
36  * W3C Recommendation for XML-Signature Syntax and Processing</a>.
37  * The XML Schema Definition is defined as:
38  * <pre><code>
39  * <lt;element name="SignatureProperties" type="ds:SignaturePropertiesType"/>
40  *   <lt;complexType name="SignaturePropertiesType">
41  *     <lt;sequence>
42  *       <lt;element ref="ds:SignatureProperty" maxOccurs="unbounded"/>
43  *     </sequence>
44  *     <lt;attribute name="Id" type="ID" use="optional"/>
45  *   </complexType>
46  * </code></pre>
47  *
48  * A <code>SignatureProperties</code> instance may be created by invoking the
49  * {@link XMLSignatureFactory#newSignatureProperties newSignatureProperties}
50  * method of the {@link XMLSignatureFactory} class; for example:
51  *
52  * <pre>
53  * XMLSignatureFactory factory = XMLSignatureFactory.getInstance("DOM");
54  * SignatureProperties properties =
55  *   factory.newSignatureProperties(props, "signature-properties-1");
56  * </pre>
57  *
58  * @author Sean Mullan
59  * @author JSR 105 Expert Group
60  * @since 1.6
61  * @see XMLSignatureFactory#newSignatureProperties(List, String)
62  * @see SignatureProperty
63  */
64 public interface SignatureProperties extends XMLStructure {
65
66     /**
67      * URI that identifies the <code>SignatureProperties</code> element (this
68      * can be specified as the value of the <code>type</code> parameter of the
69      * {@link Reference} class to identify the referent's type).
70      */
71     final static String TYPE =
72         "http://www.w3.org/2000/09/xmlsig#SignatureProperties";
73
74     /**
75      * Returns the Id of this <code>SignatureProperties</code>.
76      *
77      * @return the Id of this <code>SignatureProperties</code> (or
78      *   <code>null</code> if not specified)
79      */
80     String getId();
81
82     /**
83      * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
84      * list} of one or more {@link SignatureProperty}s that are contained in

```

```

85     * this <code>SignatureProperties</code>.
86     *
87     * @return an unmodifiable list of one or more
88     *     <code>SignatureProperty</code>s
89     */
90     @SuppressWarnings("rawtypes")
91     List getProperties();
92 }

```

## 10.7 SignatureProperty.java

Secuencia 114: javax/xml/crypto/dsig/SignatureProperty.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: SignatureProperty.java,v 1.4 2005/05/10 16:03:46 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import javax.xml.crypto.XMLStructure;
31 import java.util.List;
32
33 /**
34 * A representation of the XML <code>SignatureProperty</code> element as
35 * defined in the <a href="http://www.w3.org/TR/xmlldsig-core/">
36 * W3C Recommendation for XML-Signature Syntax and Processing</a>.
37 * The XML Schema Definition is defined as:
38 * <pre><code>
39 * <element name="SignatureProperty" type="ds:SignaturePropertyType"/>
40 *   <complexType name="SignaturePropertyType" mixed="true">
41 *     <choice maxOccurs="unbounded">

```

```

42 *      <any namespace="##other" processContents="lax"/>
43 *      <!-- (1,1) elements from (1, unbounded) namespaces -->
44 *      </choice>
45 *      <attribute name="Target" type="anyURI" use="required"/>
46 *      <attribute name="Id" type="ID" use="optional"/>
47 *      </complexType>
48 * </code></pre>
49 *
50 * A <code>SignatureProperty</code> instance may be created by invoking the
51 * {@link XMLSignatureFactory#newSignatureProperty newSignatureProperty}
52 * method of the {@link XMLSignatureFactory} class; for example:
53 *
54 * <pre>
55 * XMLSignatureFactory factory = XMLSignatureFactory.getInstance("DOM");
56 * SignatureProperty property = factory.newSignatureProperty
57 *     (Collections.singletonList(content), "#Signature-1", "TimeStamp");
58 * </pre>
59 *
60 * @author Sean Mullan
61 * @author JSR 105 Expert Group
62 * @since 1.6
63 * @see XMLSignatureFactory#newSignatureProperty(List, String, String)
64 * @see SignatureProperties
65 */
66 public interface SignatureProperty extends XMLStructure {
67
68     /**
69      * Returns the target URI of this <code>SignatureProperty</code>.
70      *
71      * @return the target URI of this <code>SignatureProperty</code> (never
72      *      <code>null</code>)
73      */
74     String getTarget();
75
76     /**
77      * Returns the Id of this <code>SignatureProperty</code>.
78      *
79      * @return the Id of this <code>SignatureProperty</code> (or
80      *      <code>null</code> if not specified)
81      */
82     String getId();
83
84     /**
85      * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
86      * list} of one or more {@link XMLStructure}s that are contained in
87      * this <code>SignatureProperty</code>. These represent additional
88      * information items concerning the generation of the {@link XMLSignature}
89      * (i.e. date/time stamp or serial numbers of cryptographic hardware used
90      * in signature generation).
91      *
92      * @return an unmodifiable list of one or more <code>XMLStructure</code>s
93      */
94     @SuppressWarnings("rawtypes")

```

```
95     List getContent();
96 }
```

## 10.8 SignedInfo.java

Secuencia 115: javax/xml/crypto/dsig/SignedInfo.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: SignedInfo.java,v 1.7 2005/05/10 16:03:47 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import javax.xml.crypto.XMLStructure;
31 import java.io.InputStream;
32 import java.util.List;
33
34 /**
35 * An representation of the XML SignedInfo element as
36 * defined in the http://www.w3.org/TR/xmlldsig-core/
37 * W3C Recommendation for XML-Signature Syntax and Processing.
38 * The XML Schema Definition is defined as:
39 * 

```
<code>
40 * <element name="SignedInfo" type="ds:SignedInfoType"/>
41 * <complexType name="SignedInfoType">
42 *   <sequence>
43 *     <element ref="ds:CanonicalizationMethod"/>
44 *     <element ref="ds:SignatureMethod"/>
45 *     <element ref="ds:Reference" maxOccurs="unbounded"/>
46 *   </sequence>
47 *   <attribute name="Id" type="ID" use="optional"/>

```


```

```

48 * <code></complexType></code>;
49 * </code></pre>
50 *
51 * A <code>SignedInfo</code> instance may be created by invoking one of the
52 * {@link XMLSignatureFactory#newSignedInfo newSignedInfo} methods of the
53 * {@link XMLSignatureFactory} class.
54 *
55 * @author Sean Mullan
56 * @author JSR 105 Expert Group
57 * @since 1.6
58 * @see XMLSignatureFactory#newSignedInfo(CanonicalizationMethod, SignatureMethod, List)
59 * @see XMLSignatureFactory#newSignedInfo(CanonicalizationMethod, SignatureMethod, List, String)
60 */
61 public interface SignedInfo extends XMLStructure {
62
63     /**
64      * Returns the canonicalization method of this <code>SignedInfo</code>.
65      *
66      * @return the canonicalization method
67      */
68     CanonicalizationMethod getCanonicalizationMethod();
69
70     /**
71      * Returns the signature method of this <code>SignedInfo</code>.
72      *
73      * @return the signature method
74      */
75     SignatureMethod getSignatureMethod();
76
77     /**
78      * Returns an {@link java.util.Collections#unmodifiableList
79      * unmodifiable list} of one or more {@link Reference}s.
80      *
81      * @return an unmodifiable list of one or more {@link Reference}s
82      */
83     @SuppressWarnings("rawtypes")
84     List getReferences();
85
86     /**
87      * Returns the optional <code>Id</code> attribute of this
88      * <code>SignedInfo</code>.
89      *
90      * @return the id (may be <code>null</code> if not specified)
91      */
92     String getId();
93
94     /**
95      * Returns the canonicalized signed info bytes after a signing or
96      * validation operation. This method is useful for debugging.
97      *
98      * @return an <code>InputStream</code> containing the canonicalized bytes,
99      *         or <code>null</code> if this <code>SignedInfo</code> has not been
100      *         signed or validated yet

```

```

101     */
102     InputStream getCanonicalizedData();
103 }

```

## 10.9 Transform.java

Secuencia 116: javax/xml/crypto/dsig/Transform.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: Transform.java,v 1.5 2005/05/10 16:03:48 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import java.io.OutputStream;
31 import java.security.spec.AlgorithmParameterSpec;
32 import javax.xml.crypto.AlgorithmMethod;
33 import javax.xml.crypto.Data;
34 import javax.xml.crypto.OctetStreamData;
35 import javax.xml.crypto.XMLCryptoContext;
36 import javax.xml.crypto.XMLStructure;
37 import javax.xml.crypto.dsig.spec.TransformParameterSpec;
38
39 /**
40 * A representation of the XML <code>Transform</code> element as
41 * defined in the <a href="http://www.w3.org/TR/xmlldsig-core/">
42 * W3C Recommendation for XML-Signature Syntax and Processing</a>.
43 * The XML Schema Definition is defined as:
44 *
45 * <pre>
46 * <element name="Transform" type="ds:TransformType"/>

```

```

47 * <complexType name="TransformType" mixed="true">
48 *   <choice minOccurs="0" maxOccurs="unbounded">
49 *     <any namespace="##other" processContents="lax"/>
50 *     <!-- (1,1) elements from (0,unbounded) namespaces -->
51 *     <element name="XPath" type="string"/>
52 *   </choice>
53 *   <attribute name="Algorithm" type="anyURI" use="required"/>
54 * </complexType>
55 * </pre>
56 *
57 * A <code>Transform</code> instance may be created by invoking the
58 * {@link XMLSignatureFactory#newTransform newTransform} method
59 * of the {@link XMLSignatureFactory} class.
60 *
61 * @author Sean Mullan
62 * @author JSR 105 Expert Group
63 * @since 1.6
64 * @see XMLSignatureFactory#newTransform(String, TransformParameterSpec)
65 */
66 public interface Transform extends XMLStructure, AlgorithmMethod {
67
68     /**
69      * The <a href="http://www.w3.org/2000/09/xmlsig#base64">Base64</a>
70      * transform algorithm URI.
71      */
72     final static String BASE64 = "http://www.w3.org/2000/09/xmlsig#base64";
73
74     /**
75      * The <a href="http://www.w3.org/2000/09/xmlsig#enveloped-signature">
76      * Enveloped Signature</a> transform algorithm URI.
77      */
78     final static String ENVELOPED =
79         "http://www.w3.org/2000/09/xmlsig#enveloped-signature";
80
81     /**
82      * The <a href="http://www.w3.org/TR/1999/REC-xpath-19991116">XPath</a>
83      * transform algorithm URI.
84      */
85     final static String XPATH = "http://www.w3.org/TR/1999/REC-xpath-19991116";
86
87     /**
88      * The <a href="http://www.w3.org/2002/06/xmlsig-filter2">
89      * XPath Filter 2</a> transform algorithm URI.
90      */
91     final static String XPATH2 = "http://www.w3.org/2002/06/xmlsig-filter2";
92
93     /**
94      * The <a href="http://www.w3.org/TR/1999/REC-xslt-19991116">XSLT</a>
95      * transform algorithm URI.
96      */
97     final static String XSLT = "http://www.w3.org/TR/1999/REC-xslt-19991116";
98
99     /**

```



```

100     * Returns the algorithm-specific input parameters associated with this
101     * <code>Transform</code>.
102     * <p>
103     * The returned parameters can be typecast to a
104     * {@link TransformParameterSpec} object.
105     *
106     * @return the algorithm-specific input parameters (may be <code>null</code>
107     *         if not specified)
108     */
109     AlgorithmParameterSpec getParameterSpec();
110
111     /**
112     * Transforms the specified data using the underlying transform algorithm.
113     *
114     * @param data the data to be transformed
115     * @param context the <code>XMLCryptoContext</code> containing
116     *        additional context (may be <code>null</code> if not applicable)
117     * @return the transformed data
118     * @throws NullPointerException if <code>data</code> is <code>null</code>
119     * @throws TransformException if an error occurs while executing the
120     *        transform
121     */
122     public abstract Data transform(Data data, XMLCryptoContext context)
123         throws TransformException;
124
125     /**
126     * Transforms the specified data using the underlying transform algorithm.
127     * If the output of this transform is an <code>OctetStreamData</code>, then
128     * this method returns <code>null</code> and the bytes are written to the
129     * specified <code>OutputStream</code>. Otherwise, the
130     * <code>OutputStream</code> is ignored and the method behaves as if
131     * {@link #transform(Data, XMLCryptoContext)} were invoked.
132     *
133     * @param data the data to be transformed
134     * @param context the <code>XMLCryptoContext</code> containing
135     *        additional context (may be <code>null</code> if not applicable)
136     * @param os the <code>OutputStream</code> that should be used to write
137     *        the transformed data to
138     * @return the transformed data (or <code>null</code> if the data was
139     *        written to the <code>OutputStream</code> parameter)
140     * @throws NullPointerException if <code>data</code> or <code>os</code>
141     *        is <code>null</code>
142     * @throws TransformException if an error occurs while executing the
143     *        transform
144     */
145     public abstract Data transform
146         (Data data, XMLCryptoContext context, OutputStream os)
147         throws TransformException;
148 }

```

## 10.10 TransformException.java

Secuencia 117: javax/xml/crypto/dsig/TransformException.java

```
1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: TransformException.java,v 1.3 2005/05/10 16:03:48 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import java.io.PrintStream;
31 import java.io.PrintWriter;
32
33 /**
34 * Indicates an exceptional condition that occurred while executing a
35 * transform algorithm.
36 *
37 * <p>A <code>TransformException</code> can contain a cause: another
38 * throwable that caused this <code>TransformException</code> to get thrown.
39 *
40 * @see Transform#transform
41 * @author Sean Mullan
42 * @author JSR 105 Expert Group
43 * @since 1.6
44 */
45 public class TransformException extends Exception {
46
47     private static final long serialVersionUID = 5082634801360427800L;
48
49     /**
50      * The throwable that caused this exception to get thrown, or null if this
51      * exception was not caused by another throwable or if the causative
52      * throwable is unknown.
53      *
```

```
54     * @serial
55     */
56     private Throwable cause;
57
58     /**
59     * Constructs a new TransformException with
60     * null as its detail message.
61     */
62     public TransformException() {
63         super();
64     }
65
66     /**
67     * Constructs a new TransformException with the specified
68     * detail message.
69     *
70     * @param message the detail message
71     */
72     public TransformException(String message) {
73         super(message);
74     }
75
76     /**
77     * Constructs a new TransformException with the
78     * specified detail message and cause.
79     * 

Note that the detail message associated with
80     * cause is not automatically incorporated in
81     * this exception's detail message.
82     *
83     * @param message the detail message
84     * @param cause the cause (A null value is permitted, and
85     *             indicates that the cause is nonexistent or unknown.)
86     */
87     public TransformException(String message, Throwable cause) {
88         super(message);
89         this.cause = cause;
90     }
91
92     /**
93     * Constructs a new TransformException with the specified
94     * cause and a detail message of
95     * (cause==null ? null : cause.toString())
96     * (which typically contains the class and detail message of
97     * cause).
98     *
99     * @param cause the cause (A null value is permitted, and
100     *             indicates that the cause is nonexistent or unknown.)
101     */
102     public TransformException(Throwable cause) {
103         super(cause==null ? null : cause.toString());
104         this.cause = cause;
105     }
106


```

```

107  /**
108   * Returns the cause of this <code>TransformException</code> or
109   * <code>null</code> if the cause is nonexistent or unknown. (The
110   * cause is the throwable that caused this
111   * <code>TransformException</code> to get thrown.)
112   *
113   * @return the cause of this <code>TransformException</code> or
114   *         <code>null</code> if the cause is nonexistent or unknown.
115   */
116  public Throwable getCause() {
117      return cause;
118  }
119
120  /**
121   * Prints this <code>TransformException</code>, its backtrace and
122   * the cause's backtrace to the standard error stream.
123   */
124  public void printStackTrace() {
125      super.printStackTrace();
126      if (cause != null) {
127          cause.printStackTrace();
128      }
129  }
130
131  /**
132   * Prints this <code>TransformException</code>, its backtrace and
133   * the cause's backtrace to the specified print stream.
134   *
135   * @param s <code>PrintStream</code> to use for output
136   */
137  public void printStackTrace(PrintStream s) {
138      super.printStackTrace(s);
139      if (cause != null) {
140          cause.printStackTrace(s);
141      }
142  }
143
144  /**
145   * Prints this <code>TransformException</code>, its backtrace and
146   * the cause's backtrace to the specified print writer.
147   *
148   * @param s <code>PrintWriter</code> to use for output
149   */
150  public void printStackTrace(PrintWriter s) {
151      super.printStackTrace(s);
152      if (cause != null) {
153          cause.printStackTrace(s);
154      }
155  }
156  }

```

## 10.11 TransformService.java

Secuencia 118: javax/xml/crypto/dsig/TransformService.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: TransformService.java,v 1.6.4.1 2005/09/15 12:42:11 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import java.security.InvalidAlgorithmParameterException;
31 import java.security.NoSuchAlgorithmException;
32 import java.security.NoSuchProviderException;
33 import java.security.Provider;
34 import java.security.Provider.Service;
35 import java.security.Security;
36 import java.util.*;
37 import javax.xml.crypto.MarshalException;
38 import javax.xml.crypto.XMLStructure;
39 import javax.xml.crypto.XMLCryptoContext;
40 import javax.xml.crypto.dsig.spec.TransformParameterSpec;
41
42 import sun.security.jca.*;
43 import sun.security.jca.GetInstance.Instance;
44
45 /**
46 * A Service Provider Interface for transform and canonicalization algorithms.
47 *
48 * <p>Each instance of <code>TransformService</code> supports a specific
49 * transform or canonicalization algorithm and XML mechanism type. To create a
50 * <code>TransformService</code>, call one of the static
51 * {@link #getInstance getInstance} methods, passing in the algorithm URI and
52 * XML mechanism type desired, for example:
```

```

53 *
54 * <blockquote><code>
55 * TransformService ts = TransformService.getInstance(Transform.XPATH2, "DOM");
56 * </code></blockquote>
57 *
58 * <p><code>TransformService</code> implementations are registered and loaded
59 * using the {@link java.security.Provider} mechanism. Each
60 * <code>TransformService</code> service provider implementation should include
61 * a <code>MechanismType</code> service attribute that identifies the XML
62 * mechanism type that it supports. If the attribute is not specified,
63 * "DOM" is assumed. For example, a service provider that supports the
64 * XPath Filter 2 Transform and DOM mechanism would be specified in the
65 * <code>Provider</code> subclass as:
66 * <pre>
67 *     put("TransformService." + Transform.XPATH2,
68 *         "org.example.XPath2TransformService");
69 *     put("TransformService." + Transform.XPATH2 + " MechanismType", "DOM");
70 * </pre>
71 * <code>TransformService</code> implementations that support the DOM
72 * mechanism type must abide by the DOM interoperability requirements defined
73 * in the
74 * <a href="../../../../technotes/guides/security/xmlsig/overview.html#DOM Mechanism
    Requirements">
75 * DOM Mechanism Requirements</a> section of the API overview. See the
76 * <a href="../../../../technotes/guides/security/xmlsig/overview.html#Service Provider">
77 * Service Providers</a> section of the API overview for a list of standard
78 * mechanism types.
79 * <p>
80 * Once a <code>TransformService</code> has been created, it can be used
81 * to process <code>Transform</code> or <code>CanonicalizationMethod</code>
82 * objects. If the <code>Transform</code> or <code>CanonicalizationMethod</code>
83 * exists in XML form (for example, when validating an existing
84 * <code>XMLSignature</code>), the {@link #init(XMLStructure, XMLCryptoContext)}
85 * method must be first called to initialize the transform and provide document
86 * context (even if there are no parameters). Alternatively, if the
87 * <code>Transform</code> or <code>CanonicalizationMethod</code> is being
88 * created from scratch, the {@link #init(TransformParameterSpec)} method
89 * is called to initialize the transform with parameters and the
90 * {@link #marshalParams marshalParams} method is called to marshal the
91 * parameters to XML and provide the transform with document context. Finally,
92 * the {@link #transform transform} method is called to perform the
93 * transformation.
94 * <p>
95 * <b>Concurrent Access</b>
96 * <p>The static methods of this class are guaranteed to be thread-safe.
97 * Multiple threads may concurrently invoke the static methods defined in this
98 * class with no ill effects.
99 *
100 * <p>However, this is not true for the non-static methods defined by this
101 * class. Unless otherwise documented by a specific provider, threads that
102 * need to access a single <code>TransformService</code> instance
103 * concurrently should synchronize amongst themselves and provide the
104 * necessary locking. Multiple threads each manipulating a different

```

```

105 * <code>TransformService</code> instance need not synchronize.
106 *
107 * @author Sean Mullan
108 * @author JSR 105 Expert Group
109 * @since 1.6
110 */
111 public abstract class TransformService implements Transform {
112
113     private String algorithm;
114     private String mechanism;
115     private Provider provider;
116
117     /**
118      * Default constructor, for invocation by subclasses.
119      */
120     protected TransformService() {}
121
122     /**
123      * Returns a <code>TransformService</code> that supports the specified
124      * algorithm URI (ex: {@link Transform#XPath2}) and mechanism type
125      * (ex: DOM).
126      *
127      * <p>This method uses the standard JCA provider lookup mechanism to
128      * locate and instantiate a <code>TransformService</code> implementation
129      * of the desired algorithm and <code>MechanismType</code> service
130      * attribute. It traverses the list of registered security
131      * <code>Provider</code>s, starting with the most preferred
132      * <code>Provider</code>. A new <code>TransformService</code> object
133      * from the first <code>Provider</code> that supports the specified
134      * algorithm and mechanism type is returned.
135      *
136      * <p>Note that the list of registered providers may be retrieved via
137      * the {@link Security#getProviders() Security.getProviders()} method.
138      *
139      * @param algorithm the URI of the algorithm
140      * @param mechanismType the type of the XML processing mechanism and
141      *     representation
142      * @return a new <code>TransformService</code>
143      * @throws NullPointerException if <code>algorithm</code> or
144      *     <code>mechanismType</code> is <code>>null</code>
145      * @throws NoSuchAlgorithmException if no <code>Provider</code> supports a
146      *     <code>TransformService</code> implementation for the specified
147      *     algorithm and mechanism type
148      * @see Provider
149      */
150     public static TransformService getInstance
151         (String algorithm, String mechanismType)
152         throws NoSuchAlgorithmException {
153         if (mechanismType == null || algorithm == null) {
154             throw new NullPointerException();
155         }
156         boolean dom = false;
157         if (mechanismType.equals("DOM")) {

```

```

158         dom = true;
159     }
160     List<Service> services = GetInstance.getServices("TransformService", algorithm);
161     for (Iterator<Service> t = services.iterator(); t.hasNext(); ) {
162         Service s = t.next();
163         String value = s.getAttribute("MechanismType");
164         if ((value == null && dom) ||
165             (value != null && value.equals(mechanismType))) {
166             Instance instance = GetInstance.getInstance(s, null);
167             TransformService ts = (TransformService) instance.impl;
168             ts.algorithm = algorithm;
169             ts.mechanism = mechanismType;
170             ts.provider = instance.provider;
171             return ts;
172         }
173     }
174     throw new NoSuchAlgorithmException
175         (algorithm + " algorithm and " + mechanismType
176          + " mechanism not available");
177 }
178
179 /**
180  * Returns a <code>TransformService</code> that supports the specified
181  * algorithm URI (ex: {@link Transform#XPath2}) and mechanism type
182  * (ex: DOM) as supplied by the specified provider. Note that the specified
183  * <code>Provider</code> object does not have to be registered in the
184  * provider list.
185  *
186  * @param algorithm the URI of the algorithm
187  * @param mechanismType the type of the XML processing mechanism and
188  *     representation
189  * @param provider the <code>Provider</code> object
190  * @return a new <code>TransformService</code>
191  * @throws NullPointerException if <code>provider</code>,
192  *     <code>algorithm</code>, or <code>mechanismType</code> is
193  *     <code>null</code>
194  * @throws NoSuchAlgorithmException if a <code>TransformService</code>
195  *     implementation for the specified algorithm and mechanism type is not
196  *     available from the specified <code>Provider</code> object
197  * @see Provider
198  */
199 public static TransformService getInstance
200     (String algorithm, String mechanismType, Provider provider)
201     throws NoSuchAlgorithmException {
202     if (mechanismType == null || algorithm == null || provider == null) {
203         throw new NullPointerException();
204     }
205
206     boolean dom = false;
207     if (mechanismType.equals("DOM")) {
208         dom = true;
209     }
210     Service s = GetInstance.getService

```



```

211     ("TransformService", algorithm, provider);
212     String value = s.getAttribute("MechanismType");
213     if ((value == null && dom) ||
214         (value != null && value.equals(mechanismType))) {
215         Instance instance = GetInstance.getInstance(s, null);
216         TransformService ts = (TransformService) instance.impl;
217         ts.algorithm = algorithm;
218         ts.mechanism = mechanismType;
219         ts.provider = instance.provider;
220         return ts;
221     }
222     throw new NoSuchAlgorithmException
223         (algorithm + " algorithm and " + mechanismType
224          + " mechanism not available");
225 }
226
227 /**
228  * Returns a TransformService that supports the specified
229  * algorithm URI (ex: {@link Transform#XPath2}) and mechanism type
230  * (ex: DOM) as supplied by the specified provider. The specified provider
231  * must be registered in the security provider list.
232  *
233  * <p>Note that the list of registered providers may be retrieved via
234  * the {@link Security#getProviders() Security.getProviders()} method.
235  *
236  * @param algorithm the URI of the algorithm
237  * @param mechanismType the type of the XML processing mechanism and
238  *     representation
239  * @param provider the string name of the provider
240  * @return a new TransformService
241  * @throws NoSuchProviderException if the specified provider is not
242  *     registered in the security provider list
243  * @throws NullPointerException if provider,
244  *     mechanismType, or algorithm is
245  *     null
246  * @throws NoSuchAlgorithmException if a TransformService
247  *     implementation for the specified algorithm and mechanism type is not
248  *     available from the specified provider
249  * @see Provider
250  */
251 public static TransformService getInstance
252     (String algorithm, String mechanismType, String provider)
253     throws NoSuchAlgorithmException, NoSuchProviderException {
254     if (mechanismType == null || algorithm == null || provider == null) {
255         throw new NullPointerException();
256     } else if (provider.length() == 0) {
257         throw new NoSuchProviderException();
258     }
259     boolean dom = false;
260     if (mechanismType.equals("DOM")) {
261         dom = true;
262     }
263     Service s = GetInstance.getService

```

```

264         ("TransformService", algorithm, provider);
265     String value = s.getAttribute("MechanismType");
266     if ((value == null && dom) ||
267         (value != null && value.equals(mechanismType))) {
268         Instance instance = GetInstance.getInstance(s, null);
269         TransformService ts = (TransformService) instance.impl;
270         ts.algorithm = algorithm;
271         ts.mechanism = mechanismType;
272         ts.provider = instance.provider;
273         return ts;
274     }
275     throw new NoSuchAlgorithmException
276         (algorithm + " algorithm and " + mechanismType
277          + " mechanism not available");
278 }
279
280 private static class MechanismMapEntry implements Map.Entry<String,String> {
281     private final String mechanism;
282     private final String algorithm;
283     private final String key;
284     MechanismMapEntry(String algorithm, String mechanism) {
285         this.algorithm = algorithm;
286         this.mechanism = mechanism;
287         this.key = "TransformService." + algorithm + " MechanismType";
288     }
289     public boolean equals(Object o) {
290         if (!(o instanceof Map.Entry)) {
291             return false;
292         }
293         Map.Entry<?,?> e = (Map.Entry<?,?>) o;
294         return (getKey()==null ?
295             e.getKey()==null : getKey().equals(e.getKey())) &&
296             (getValue()==null ?
297             e.getValue()==null : getValue().equals(e.getValue()));
298     }
299     public String getKey() {
300         return key;
301     }
302     public String getValue() {
303         return mechanism;
304     }
305     public String setValue(String value) {
306         throw new UnsupportedOperationException();
307     }
308     public int hashCode() {
309         return (getKey()==null ? 0 : getKey().hashCode()) ^
310             (getValue()==null ? 0 : getValue().hashCode());
311     }
312 }
313
314 /**
315  * Returns the mechanism type supported by this <code>TransformService</code>.
316  *

```

```

317     * @return the mechanism type
318     */
319     public final String getMechanismType() {
320         return mechanism;
321     }
322
323     /**
324     * Returns the URI of the algorithm supported by this
325     * <code>TransformService</code>.
326     *
327     * @return the algorithm URI
328     */
329     public final String getAlgorithm() {
330         return algorithm;
331     }
332
333     /**
334     * Returns the provider of this <code>TransformService</code>.
335     *
336     * @return the provider
337     */
338     public final Provider getProvider() {
339         return provider;
340     }
341
342     /**
343     * Initializes this <code>TransformService</code> with the specified
344     * parameters.
345     *
346     * <p>If the parameters exist in XML form, the
347     * {@link #init(XMLStructure, XMLCryptoContext)} method should be used to
348     * initialize the <code>TransformService</code>.
349     *
350     * @param params the algorithm parameters (may be <code>null</code> if
351     *     not required or optional)
352     * @throws InvalidAlgorithmParameterException if the specified parameters
353     *     are invalid for this algorithm
354     */
355     public abstract void init(TransformParameterSpec params)
356         throws InvalidAlgorithmParameterException;
357
358     /**
359     * Marshals the algorithm-specific parameters. If there are no parameters
360     * to be marshalled, this method returns without throwing an exception.
361     *
362     * @param parent a mechanism-specific structure containing the parent
363     *     node that the marshalled parameters should be appended to
364     * @param context the <code>XMLCryptoContext</code> containing
365     *     additional context (may be <code>null</code> if not applicable)
366     * @throws ClassCastException if the type of <code>parent</code> or
367     *     <code>context</code> is not compatible with this
368     *     <code>TransformService</code>
369     * @throws NullPointerException if <code>parent</code> is <code>null</code>

```

```

370     * @throws MarshalException if the parameters cannot be marshalled
371     */
372     public abstract void marshalParams
373         (XMLStructure parent, XMLCryptoContext context)
374         throws MarshalException;
375
376     /**
377     * Initializes this <code>TransformService</code> with the specified
378     * parameters and document context.
379     *
380     * @param parent a mechanism-specific structure containing the parent
381     *     structure
382     * @param context the <code>XMLCryptoContext</code> containing
383     *     additional context (may be <code>null</code> if not applicable)
384     * @throws ClassCastException if the type of <code>parent</code> or
385     *     <code>context</code> is not compatible with this
386     *     <code>TransformService</code>
387     * @throws NullPointerException if <code>parent</code> is <code>null</code>
388     * @throws InvalidAlgorithmParameterException if the specified parameters
389     *     are invalid for this algorithm
390     */
391     public abstract void init(XMLStructure parent, XMLCryptoContext context)
392         throws InvalidAlgorithmParameterException;
393 }

```

## 10.12 XMLObject.java

Secuencia 119: javax/xml/crypto/dsig/XMLObject.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25

```

```

26  /*
27  * =====
28  *
29  * (C) Copyright IBM Corp. 2003 All Rights Reserved.
30  *
31  * =====
32  */
33  /*
34  * $Id: XMLObject.java,v 1.5 2005/05/10 16:03:48 mullan Exp $
35  */
36  package javax.xml.crypto.dsig;
37
38  import java.util.List;
39  import javax.xml.crypto.XMLStructure;
40
41  /**
42   * A representation of the XML <code>Object</code> element as defined in
43   * the http://www.w3.org/TR/xmlsig-core/
44   * W3C Recommendation for XML-Signature Syntax and Processing.
45   * An <code>XMLObject</code> may contain any data and may include optional
46   * MIME type, ID, and encoding attributes. The XML Schema Definition is
47   * defined as:
48   *
49   * 

```
<code>
50   * <lt;element name="Object" type="ds:ObjectType"/>
51   * <lt;complexType name="ObjectType" mixed="true">
52   *   <lt;sequence minOccurs="0" maxOccurs="unbounded">
53   *     <lt;any namespace="##any" processContents="lax"/>
54   *   <lt;/sequence>
55   *   <lt;attribute name="Id" type="ID" use="optional"/>
56   *   <lt;attribute name="MimeType" type="string" use="optional"/>
57   *   <lt;attribute name="Encoding" type="anyURI" use="optional"/>
58   * <lt;/complexType>
59   * </code></pre>
60   *
61   * A <code>XMLObject</code> instance may be created by invoking the
62   * {@link XMLSignatureFactory#newXMLObject newXMLObject} method of the
63   * {@link XMLSignatureFactory} class; for example:
64   *
65   * 

```
<code>
66   * XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
67   * List content = Collections.singletonList(fac.newManifest(references));
68   * XMLObject object = factory.newXMLObject(content, "object-1", null, null);
69   * </pre>
70   *
71   * <p>Note that this class is named <code>XMLObject</code> rather than
72   * <code>Object</code> to avoid naming clashes with the existing
73   * {@link java.lang.Object java.lang.Object} class.
74   *
75   * @author Sean Mullan
76   * @author JSR 105 Expert Group
77   * @author Joyce L. Leung
78   * @since 1.6

```


```


```

```

79  * @see XMLSignatureFactory#newXMLObject(List, String, String, String)
80  */
81  public interface XMLObject extends XMLStructure {
82
83      /**
84       * URI that identifies the <code>Object</code> element (this can be
85       * specified as the value of the <code>type</code> parameter of the
86       * {@link Reference} class to identify the referent's type).
87       */
88       final static String TYPE = "http://www.w3.org/2000/09/xmldsig#Object";
89
90      /**
91       * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
92       * list} of {@link XMLStructure}s contained in this <code>XMLObject</code>,
93       * which represent elements from any namespace.
94       *
95       * <p>If there is a public subclass representing the type of
96       * <code>XMLStructure</code>, it is returned as an instance of that class
97       * (ex: a <code>SignatureProperties</code> element would be returned
98       * as an instance of {@link javax.xml.crypto.dsig.SignatureProperties}).
99       *
100      * @return an unmodifiable list of <code>XMLStructure</code>s (may be empty
101      *      but never <code>null</code>)
102      */
103      @SuppressWarnings("rawtypes")
104      List getContent();
105
106      /**
107       * Returns the Id of this <code>XMLObject</code>.
108       *
109       * @return the Id (or <code>null</code> if not specified)
110       */
111      String getId();
112
113      /**
114       * Returns the mime type of this <code>XMLObject</code>. The
115       * mime type is an optional attribute which describes the data within this
116       * <code>XMLObject</code> (independent of its encoding).
117       *
118       * @return the mime type (or <code>null</code> if not specified)
119       */
120      String getMimeType();
121
122      /**
123       * Returns the encoding URI of this <code>XMLObject</code>. The encoding
124       * URI identifies the method by which the object is encoded.
125       *
126       * @return the encoding URI (or <code>null</code> if not specified)
127       */
128      String getEncoding();
129  }

```

## 10.13 XMLSignContext.java

Secuencia 120: javax/xml/crypto/dsig/XMLSignContext.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: XMLSignContext.java,v 1.8 2005/05/10 16:03:48 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import javax.xml.crypto.KeySelector;
31 import javax.xml.crypto.XMLCryptoContext;
32
33 /**
34 * Contains context information for generating XML Signatures. This interface
35 * is primarily intended for type-safety.
36 *
37 * <p>Note that <code>XMLSignContext</code> instances can contain
38 * information and state specific to the XML signature structure it is
39 * used with. The results are unpredictable if an
40 * <code>XMLSignContext</code> is used with different signature structures
41 * (for example, you should not use the same <code>XMLSignContext</code>
42 * instance to sign two different {@link XMLSignature} objects).
43 * <p>
44 * <b><a name="Supported Properties"></a>Supported Properties</b>
45 * <p>The following properties can be set using the
46 * {@link #setProperty setProperty} method.
47 * <ul>
48 * <li><code>javax.xml.crypto.dsig.cacheReference</code>: value must be a
49 *     {@link Boolean}. This property controls whether or not the digested
50 *     {@link Reference} objects will cache the dereferenced content and

```

```

51 *     pre-digested input for subsequent retrieval via the
52 *     {@link Reference#getDereferencedData Reference.getDereferencedData} and
53 *     {@link Reference#getDigestInputStream Reference.getDigestInputStream}
54 *     methods. The default value if not specified is
55 *     Boolean.FALSE.
56 * </ul>
57 *
58 * @author Sean Mullan
59 * @author JSR 105 Expert Group
60 * @since 1.6
61 * @see XMLSignature#sign(XMLSignContext)
62 */
63 public interface XMLSignContext extends XMLCryptoContext {}

```

## 10.14 XMLSignature.java

Secuencia 121: javax/xml/crypto/dsig/XMLSignature.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 /*
27 * =====
28 *
29 * (C) Copyright IBM Corp. 2003 All Rights Reserved.
30 *
31 * =====
32 */
33 /*
34 * $Id: XMLSignature.java,v 1.10 2005/05/10 16:03:48 mullan Exp $
35 */
36 package javax.xml.crypto.dsig;

```



```

37
38 import javax.xml.crypto.KeySelector;
39 import javax.xml.crypto.KeySelectorResult;
40 import javax.xml.crypto.MarshalException;
41 import javax.xml.crypto.XMLStructure;
42 import javax.xml.crypto.dsig.keyinfo.KeyInfo;
43 import java.security.Signature;
44 import java.util.List;
45
46 /**
47  * A representation of the XML Signature element as
48  * defined in the http://www.w3.org/TR/xmlsig-core/
49  * W3C Recommendation for XML-Signature Syntax and Processing.
50  * This class contains methods for signing and validating XML signatures
51  * with behavior as defined by the W3C specification. The XML Schema Definition
52  * is defined as:
53  * <pre><code>
54  * <element name="Signature" type="ds:SignatureType"/>
55  * <complexType name="SignatureType">
56  *   <sequence>
57  *     <element ref="ds:SignedInfo"/>
58  *     <element ref="ds:SignatureValue"/>
59  *     <element ref="ds:KeyInfo" minOccurs="0"/>
60  *     <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/>
61  *   </sequence>
62  *   <attribute name="Id" type="ID" use="optional"/>
63  * </complexType>
64  * </code></pre>
65  * <p>
66  * An XMLSignature instance may be created by invoking one of the
67  * {@link XMLSignatureFactory#newXMLSignature newXMLSignature} methods of the
68  * {@link XMLSignatureFactory} class.
69  *
70  * <p>If the contents of the underlying document containing the
71  * XMLSignature are subsequently modified, the behavior is
72  * undefined.
73  *
74  * <p>Note that this class is named XMLSignature rather than
75  * Signature to avoid naming clashes with the existing
76  * {@link Signature java.security.Signature} class.
77  *
78  * @see XMLSignatureFactory#newXMLSignature(SignedInfo, KeyInfo)
79  * @see XMLSignatureFactory#newXMLSignature(SignedInfo, KeyInfo, List, String, String)
80  * @author Joyce L. Leung
81  * @author Sean Mullan
82  * @author Erwin van der Koogh
83  * @author JSR 105 Expert Group
84  * @since 1.6
85  */
86 public interface XMLSignature extends XMLStructure {
87
88     /**
89      * The XML Namespace URI of the W3C Recommendation for XML-Signature

```

```

90     * Syntax and Processing.
91     */
92     final static String XMLNS = "http://www.w3.org/2000/09/xmlsig#";
93
94     /**
95      * Validates the signature according to the
96      * <a href="http://www.w3.org/TR/xmlsig-core/#sec-CoreValidation">
97      * core validation processing rules</a>. This method validates the
98      * signature using the existing state, it does not unmarshal and
99      * reinitialize the contents of the <code>XMLSignature</code> using the
100     * location information specified in the context.
101     *
102     * <p>This method only validates the signature the first time it is
103     * invoked. On subsequent invocations, it returns a cached result.
104     *
105     * @param validateContext the validating context
106     * @return <code>true</code> if the signature passed core validation,
107     *         otherwise <code>false</code>
108     * @throws ClassCastException if the type of <code>validateContext</code>
109     *         is not compatible with this <code>XMLSignature</code>
110     * @throws NullPointerException if <code>validateContext</code> is
111     *         <code>null</code>
112     * @throws XMLSignatureException if an unexpected error occurs during
113     *         validation that prevented the validation operation from completing
114     */
115     boolean validate(XMLValidateContext validateContext)
116         throws XMLSignatureException;
117
118     /**
119      * Returns the key info of this <code>XMLSignature</code>.
120      *
121      * @return the key info (may be <code>null</code> if not specified)
122      */
123     KeyInfo getKeyInfo();
124
125     /**
126      * Returns the signed info of this <code>XMLSignature</code>.
127      *
128      * @return the signed info (never <code>null</code>)
129      */
130     SignedInfo getSignedInfo();
131
132     /**
133      * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
134      * list} of {@link XMLObject}s contained in this <code>XMLSignature</code>.
135      *
136      * @return an unmodifiable list of <code>XMLObject</code>s (may be empty
137      *         but never <code>null</code>)
138      */
139     @SuppressWarnings("rawtypes")
140     List getObjects();
141
142     /**

```

```

143     * Returns the optional Id of this <code>XMLSignature</code>.
144     *
145     * @return the Id (may be <code>null</code> if not specified)
146     */
147     String getId();
148
149     /**
150     * Returns the signature value of this <code>XMLSignature</code>.
151     *
152     * @return the signature value
153     */
154     SignatureValue getSignatureValue();
155
156     /**
157     * Signs this <code>XMLSignature</code>.
158     *
159     * <p>If this method throws an exception, this <code>XMLSignature</code> and
160     * the <code>signContext</code> parameter will be left in the state that
161     * it was in prior to the invocation.
162     *
163     * @param signContext the signing context
164     * @throws ClassCastException if the type of <code>signContext</code> is
165     *     not compatible with this <code>XMLSignature</code>
166     * @throws NullPointerException if <code>signContext</code> is
167     *     <code>null</code>
168     * @throws MarshalException if an exception occurs while marshalling
169     * @throws XMLSignatureException if an unexpected exception occurs while
170     *     generating the signature
171     */
172     void sign(XMLSignContext signContext) throws MarshalException,
173         XMLSignatureException;
174
175     /**
176     * Returns the result of the {@link KeySelector}, if specified, after
177     * this <code>XMLSignature</code> has been signed or validated.
178     *
179     * @return the key selector result, or <code>null</code> if a key
180     *     selector has not been specified or this <code>XMLSignature</code>
181     *     has not been signed or validated
182     */
183     KeySelectorResult getKeySelectorResult();
184
185     /**
186     * A representation of the XML <code>SignatureValue</code> element as
187     * defined in the <a href="http://www.w3.org/TR/xmlsig-core/">
188     * W3C Recommendation for XML-Signature Syntax and Processing</a>.
189     * The XML Schema Definition is defined as:
190     * <p>
191     * <pre>
192     *     <element name="SignatureValue" type="ds:SignatureValueType"/>
193     *     <complexType name="SignatureValueType">
194     *         <simpleContent>
195     *             <extension base="base64Binary"/>

```



## 10.15 XMLSignatureException.java

Secuencia 122: javax/xml/crypto/dsig/XMLSignatureException.java

```
1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: XMLSignatureException.java,v 1.5 2005/05/10 16:03:48 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import java.io.PrintStream;
31 import java.io.PrintWriter;
32
33 /**
34 * Indicates an exceptional condition that occurred during the XML
35 * signature generation or validation process.
36 *
37 * <p>An <code>XMLSignatureException</code> can contain a cause: another
38 * throwable that caused this <code>XMLSignatureException</code> to get thrown.
39 *
40 * @since 1.6
41 */
42 public class XMLSignatureException extends Exception {
43
44     private static final long serialVersionUID = -3438102491013869995L;
45
46     /**
47      * The throwable that caused this exception to get thrown, or null if this
48      * exception was not caused by another throwable or if the causative
49      * throwable is unknown.
50      */
51 }
```

```
51     * @serial
52     */
53     private Throwable cause;
54
55     /**
56     * Constructs a new XMLSignatureException with
57     * null as its detail message.
58     */
59     public XMLSignatureException() {
60         super();
61     }
62
63     /**
64     * Constructs a new XMLSignatureException with the specified
65     * detail message.
66     *
67     * @param message the detail message
68     */
69     public XMLSignatureException(String message) {
70         super(message);
71     }
72
73     /**
74     * Constructs a new XMLSignatureException with the
75     * specified detail message and cause.
76     * 

Note that the detail message associated with
77     * cause is not automatically incorporated in
78     * this exception's detail message.
79     *
80     * @param message the detail message
81     * @param cause the cause (A null value is permitted, and
82     *             indicates that the cause is nonexistent or unknown.)
83     */
84     public XMLSignatureException(String message, Throwable cause) {
85         super(message);
86         this.cause = cause;
87     }
88
89     /**
90     * Constructs a new XMLSignatureException with the specified
91     * cause and a detail message of
92     * (cause==null ? null : cause.toString())
93     * (which typically contains the class and detail message of
94     * cause).
95     *
96     * @param cause the cause (A null value is permitted, and
97     *             indicates that the cause is nonexistent or unknown.)
98     */
99     public XMLSignatureException(Throwable cause) {
100         super(cause==null ? null : cause.toString());
101         this.cause = cause;
102     }
103


```

```

104  /**
105   * Returns the cause of this <code>XMLSignatureException</code> or
106   * <code>null</code> if the cause is nonexistent or unknown. (The
107   * cause is the throwable that caused this
108   * <code>XMLSignatureException</code> to get thrown.)
109   *
110   * @return the cause of this <code>XMLSignatureException</code> or
111   *         <code>null</code> if the cause is nonexistent or unknown.
112   */
113  public Throwable getCause() {
114      return cause;
115  }
116
117  /**
118   * Prints this <code>XMLSignatureException</code>, its backtrace and
119   * the cause's backtrace to the standard error stream.
120   */
121  public void printStackTrace() {
122      super.printStackTrace();
123      if (cause != null) {
124          cause.printStackTrace();
125      }
126  }
127
128  /**
129   * Prints this <code>XMLSignatureException</code>, its backtrace and
130   * the cause's backtrace to the specified print stream.
131   *
132   * @param s <code>PrintStream</code> to use for output
133   */
134  public void printStackTrace(PrintStream s) {
135      super.printStackTrace(s);
136      if (cause != null) {
137          cause.printStackTrace(s);
138      }
139  }
140
141  /**
142   * Prints this <code>XMLSignatureException</code>, its backtrace and
143   * the cause's backtrace to the specified print writer.
144   *
145   * @param s <code>PrintWriter</code> to use for output
146   */
147  public void printStackTrace(PrintWriter s) {
148      super.printStackTrace(s);
149      if (cause != null) {
150          cause.printStackTrace(s);
151      }
152  }
153 }

```

Secuencia 123: javax/xml/crypto/dsig/XMLSignatureFactory.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: XMLSignatureFactory.java,v 1.14 2005/09/15 14:29:01 mullan Exp $
27 */
28 package javax.xml.crypto.dsig;
29
30 import javax.xml.crypto.Data;
31 import javax.xml.crypto.MarshalException;
32 import javax.xml.crypto.NoSuchMechanismException;
33 import javax.xml.crypto.URIDereferencer;
34 import javax.xml.crypto.XMLStructure;
35 import javax.xml.crypto.dom.DOMStructure;
36 import javax.xml.crypto.dsig.keyinfo.KeyInfo;
37 import javax.xml.crypto.dsig.keyinfo.KeyInfoFactory;
38 import javax.xml.crypto.dsig.spec.*;
39 import javax.xml.crypto.dsig.dom.DOMValidateContext;
40 import javax.xml.crypto.dsig.dom.DOMSignContext;
41
42 import java.security.InvalidAlgorithmParameterException;
43 import java.security.NoSuchAlgorithmException;
44 import java.security.NoSuchProviderException;
45 import java.security.Provider;
46 import java.security.Security;
47 import java.util.List;
48
49 import sun.security.jca.*;
50 import sun.security.jca.GetInstance.Instance;
51
52 /**
```



```

53 * A factory for creating {@link XMLSignature} objects from scratch or
54 * for unmarshalling an <code>XMLSignature</code> object from a corresponding
55 * XML representation.
56 *
57 * <h2>XMLSignatureFactory Type</h2>
58 *
59 * <p>Each instance of <code>XMLSignatureFactory</code> supports a specific
60 * XML mechanism type. To create an <code>XMLSignatureFactory</code>, call one
61 * of the static {@link #getInstance getInstance} methods, passing in the XML
62 * mechanism type desired, for example:
63 *
64 * <blockquote><code>
65 * XMLSignatureFactory factory = XMLSignatureFactory.getInstance("DOM");
66 * </code></blockquote>
67 *
68 * <p>The objects that this factory produces will be based
69 * on DOM and abide by the DOM interoperability requirements as defined in the
70 * <a href="../../../../technotes/guides/security/xmlsig/overview.html#DOM Mechanism
    Requirements">
71 * DOM Mechanism Requirements</a> section of the API overview. See the
72 * <a href="../../../../technotes/guides/security/xmlsig/overview.html#Service Provider">
73 * Service Providers</a> section of the API overview for a list of standard
74 * mechanism types.
75 *
76 * <p><code>XMLSignatureFactory</code> implementations are registered and loaded
77 * using the {@link java.security.Provider} mechanism.
78 * For example, a service provider that supports the
79 * DOM mechanism would be specified in the <code>Provider</code> subclass as:
80 * <pre>
81 *     put("XMLSignatureFactory.DOM", "org.example.DOMXMLSignatureFactory");
82 * </pre>
83 *
84 * <p>An implementation MUST minimally support the default mechanism type: DOM.
85 *
86 * <p>Note that a caller must use the same <code>XMLSignatureFactory</code>
87 * instance to create the <code>XMLStructure</code>s of a particular
88 * <code>XMLSignature</code> that is to be generated. The behavior is
89 * undefined if <code>XMLStructure</code>s from different providers or
90 * different mechanism types are used together.
91 *
92 * <p>Also, the <code>XMLStructure</code>s that are created by this factory
93 * may contain state specific to the <code>XMLSignature</code> and are not
94 * intended to be reusable.
95 *
96 * <h2>Creating XMLSignatures from scratch</h2>
97 *
98 * <p>Once the <code>XMLSignatureFactory</code> has been created, objects
99 * can be instantiated by calling the appropriate method. For example, a
100 * {@link Reference} instance may be created by invoking one of the
101 * {@link #newReference newReference} methods.
102 *
103 * <h2>Unmarshalling XMLSignatures from XML</h2>
104 *

```

```

105 * <p>Alternatively, an XMLSignature may be created from an
106 * existing XML representation by invoking the {@link #unmarshalXMLSignature
107 * unmarshalXMLSignature} method and passing it a mechanism-specific
108 * {@link XMLValidateContext} instance containing the XML content:
109 *
110 * <pre>
111 * DOMValidateContext context = new DOMValidateContext(key, signatureElement);
112 * XMLSignature signature = factory.unmarshalXMLSignature(context);
113 * </pre>
114 *
115 * Each XMLSignatureFactory must support the required
116 * XMLValidateContext types for that factory type, but may support
117 * others. A DOM XMLSignatureFactory must support {@link
118 * DOMValidateContext} objects.
119 *
120 * <h2>Signing and marshalling XMLSignatures to XML</h2>
121 *
122 * Each XMLSignature created by the factory can also be
123 * marshalled to an XML representation and signed, by invoking the
124 * {@link XMLSignature#sign sign} method of the
125 * {@link XMLSignature} object and passing it a mechanism-specific
126 * {@link XMLSignContext} object containing the signing key and
127 * marshalling parameters (see {@link DOMSignContext}).
128 * For example:
129 *
130 * <pre>
131 * DOMSignContext context = new DOMSignContext(privateKey, document);
132 * signature.sign(context);
133 * </pre>
134 *
135 * <b>Concurrent Access</b>
136 * <p>The static methods of this class are guaranteed to be thread-safe.
137 * Multiple threads may concurrently invoke the static methods defined in this
138 * class with no ill effects.
139 *
140 * <p>However, this is not true for the non-static methods defined by this
141 * class. Unless otherwise documented by a specific provider, threads that
142 * need to access a single XMLSignatureFactory instance
143 * concurrently should synchronize amongst themselves and provide the
144 * necessary locking. Multiple threads each manipulating a different
145 * XMLSignatureFactory instance need not synchronize.
146 *
147 * @author Sean Mullan
148 * @author JSR 105 Expert Group
149 * @since 1.6
150 */
151 public abstract class XMLSignatureFactory {
152
153     private String mechanismType;
154     private Provider provider;
155
156     /**
157      * Default constructor, for invocation by subclasses.

```

```

158     */
159     protected XMLSignatureFactory() {}
160
161     /**
162      * Returns an XMLSignatureFactory that supports the
163      * specified XML processing mechanism and representation type (ex: "DOM").
164      *
165      * <p>This method uses the standard JCA provider lookup mechanism to
166      * locate and instantiate an XMLSignatureFactory
167      * implementation of the desired mechanism type. It traverses the list of
168      * registered security Providers, starting with the most
169      * preferred Provider. A new XMLSignatureFactory
170      * object from the first Provider that supports the specified
171      * mechanism is returned.
172      *
173      * <p>Note that the list of registered providers may be retrieved via
174      * the {@link Security#getProviders() Security.getProviders()} method.
175      *
176      * @param mechanismType the type of the XML processing mechanism and
177      * representation. See the <a
178      * href="../../../../technotes/guides/security/xmlsig/overview.html#Service Provider">
179      * Service Providers</a> section of the API overview for a list of
180      * standard mechanism types.
181      * @return a new XMLSignatureFactory
182      * @throws NullPointerException if mechanismType is
183      * <code>null</code>
184      * @throws NoSuchMechanismException if no Provider supports an
185      * XMLSignatureFactory implementation for the specified
186      * mechanism
187      * @see Provider
188      */
189     public static XMLSignatureFactory getInstance(String mechanismType) {
190         if (mechanismType == null) {
191             throw new NullPointerException("mechanismType cannot be null");
192         }
193         Instance instance;
194         try {
195             instance = GetInstance.getInstance
196                 ("XMLSignatureFactory", null, mechanismType);
197         } catch (NoSuchAlgorithmException nsae) {
198             throw new NoSuchMechanismException(nsae);
199         }
200         XMLSignatureFactory factory = (XMLSignatureFactory) instance.impl;
201         factory.mechanismType = mechanismType;
202         factory.provider = instance.provider;
203         return factory;
204     }
205
206     /**
207      * Returns an XMLSignatureFactory that supports the
208      * requested XML processing mechanism and representation type (ex: "DOM"),
209      * as supplied by the specified provider. Note that the specified
210      * Provider object does not have to be registered in the

```

```

211 * provider list.
212 *
213 * @param mechanismType the type of the XML processing mechanism and
214 * representation. See the <a
215 * href="../../../../technotes/guides/security/xmldsig/overview.html#Service Provider">
216 * Service Providers</a> section of the API overview for a list of
217 * standard mechanism types.
218 * @param provider the <code>Provider</code> object
219 * @return a new <code>XMLSignatureFactory</code>
220 * @throws NullPointerException if <code>provider</code> or
221 * <code>mechanismType</code> is <code>null</code>
222 * @throws NoSuchAlgorithmException if an <code>XMLSignatureFactory</code>
223 * implementation for the specified mechanism is not available
224 * from the specified <code>Provider</code> object
225 * @see Provider
226 */
227 public static XMLSignatureFactory getInstance(String mechanismType,
228     Provider provider) {
229     if (mechanismType == null) {
230         throw new NullPointerException("mechanismType cannot be null");
231     } else if (provider == null) {
232         throw new NullPointerException("provider cannot be null");
233     }
234
235     Instance instance;
236     try {
237         instance = GetInstance.getInstance
238             ("XMLSignatureFactory", null, mechanismType, provider);
239     } catch (NoSuchAlgorithmException nsae) {
240         throw new NoSuchAlgorithmException(nsae);
241     }
242     XMLSignatureFactory factory = (XMLSignatureFactory) instance.impl;
243     factory.mechanismType = mechanismType;
244     factory.provider = instance.provider;
245     return factory;
246 }
247
248 /**
249 * Returns an <code>XMLSignatureFactory</code> that supports the
250 * requested XML processing mechanism and representation type (ex: "DOM"),
251 * as supplied by the specified provider. The specified provider must be
252 * registered in the security provider list.
253 *
254 * <p>Note that the list of registered providers may be retrieved via
255 * the {@link Security#getProviders() Security.getProviders()} method.
256 *
257 * @param mechanismType the type of the XML processing mechanism and
258 * representation. See the <a
259 * href="../../../../technotes/guides/security/xmldsig/overview.html#Service Provider">
260 * Service Providers</a> section of the API overview for a list of
261 * standard mechanism types.
262 * @param provider the string name of the provider
263 * @return a new <code>XMLSignatureFactory</code>

```

```

264 * @throws NoSuchProviderException if the specified provider is not
265 *   registered in the security provider list
266 * @throws NullPointerException if <code>provider</code> or
267 *   <code>mechanismType</code> is <code>null</code>
268 * @throws NoSuchMechanismException if an <code>XMLSignatureFactory</code>
269 *   implementation for the specified mechanism is not
270 *   available from the specified provider
271 * @see Provider
272 */
273 public static XMLSignatureFactory getInstance(String mechanismType,
274   String provider) throws NoSuchProviderException {
275     if (mechanismType == null) {
276         throw new NullPointerException("mechanismType cannot be null");
277     } else if (provider == null) {
278         throw new NullPointerException("provider cannot be null");
279     } else if (provider.length() == 0) {
280         throw new NoSuchProviderException();
281     }
282
283     Instance instance;
284     try {
285         instance = GetInstance.getInstance
286             ("XMLSignatureFactory", null, mechanismType, provider);
287     } catch (NoSuchAlgorithmException nsae) {
288         throw new NoSuchMechanismException(nsae);
289     }
290     XMLSignatureFactory factory = (XMLSignatureFactory) instance.impl;
291     factory.mechanismType = mechanismType;
292     factory.provider = instance.provider;
293     return factory;
294 }
295
296 /**
297  * Returns an <code>XMLSignatureFactory</code> that supports the
298  * default XML processing mechanism and representation type ("DOM").
299  *
300  * <p>This method uses the standard JCA provider lookup mechanism to
301  * locate and instantiate an <code>XMLSignatureFactory</code>
302  * implementation of the default mechanism type. It traverses the list of
303  * registered security <code>Provider</code>s, starting with the most
304  * preferred <code>Provider</code>. A new <code>XMLSignatureFactory</code>
305  * object from the first <code>Provider</code> that supports the DOM
306  * mechanism is returned.
307  *
308  * <p>Note that the list of registered providers may be retrieved via
309  * the {@link Security#getProviders() Security.getProviders()} method.
310  *
311  * @return a new <code>XMLSignatureFactory</code>
312  * @throws NoSuchMechanismException if no <code>Provider</code> supports an
313  *   <code>XMLSignatureFactory</code> implementation for the DOM
314  *   mechanism
315  * @see Provider
316 */

```

```

317     public static XMLSignatureFactory getInstance() {
318         return getInstance("DOM");
319     }
320
321     /**
322      * Returns the type of the XML processing mechanism and representation
323      * supported by this <code>XMLSignatureFactory</code> (ex: "DOM").
324      *
325      * @return the XML processing mechanism type supported by this
326      *         <code>XMLSignatureFactory</code>
327      */
328     public final String getMechanismType() {
329         return mechanismType;
330     }
331
332     /**
333      * Returns the provider of this <code>XMLSignatureFactory</code>.
334      *
335      * @return the provider of this <code>XMLSignatureFactory</code>
336      */
337     public final Provider getProvider() {
338         return provider;
339     }
340
341     /**
342      * Creates an <code>XMLSignature</code> and initializes it with the contents
343      * of the specified <code>SignedInfo</code> and <code>KeyInfo</code>
344      * objects.
345      *
346      * @param si the signed info
347      * @param ki the key info (may be <code>null</code>)
348      * @return an <code>XMLSignature</code>
349      * @throws NullPointerException if <code>si</code> is <code>null</code>
350      */
351     public abstract XMLSignature newXMLSignature(SignedInfo si, KeyInfo ki);
352
353     /**
354      * Creates an <code>XMLSignature</code> and initializes it with the
355      * specified parameters.
356      *
357      * @param si the signed info
358      * @param ki the key info (may be <code>null</code>)
359      * @param objects a list of {@link XMLObject}s (may be empty or
360      *              <code>null</code>)
361      * @param id the Id (may be <code>null</code>)
362      * @param signatureValueId the SignatureValue Id (may be <code>null</code>)
363      * @return an <code>XMLSignature</code>
364      * @throws NullPointerException if <code>si</code> is <code>null</code>
365      * @throws ClassCastException if any of the <code>objects</code> are not of
366      *         type <code>XMLObject</code>
367      */
368     @SuppressWarnings("rawtypes")
369     public abstract XMLSignature newXMLSignature(SignedInfo si, KeyInfo ki,

```

```

370     List objects, String id, String signatureValueId);
371
372     /**
373      * Creates a Reference with the specified URI and digest
374      * method.
375      *
376      * @param uri the reference URI (may be null)
377      * @param dm the digest method
378      * @return a Reference
379      * @throws IllegalArgumentException if uri is not RFC 2396
380      *         compliant
381      * @throws NullPointerException if dm is null
382      */
383     public abstract Reference newReference(String uri, DigestMethod dm);
384
385     /**
386      * Creates a Reference with the specified parameters.
387      *
388      * @param uri the reference URI (may be null)
389      * @param dm the digest method
390      * @param transforms a list of {@link Transform}s. The list is defensively
391      *         copied to protect against subsequent modification. May be
392      *         null or empty.
393      * @param type the reference type, as a URI (may be null)
394      * @param id the reference ID (may be null)
395      * @return a Reference
396      * @throws ClassCastException if any of the transforms are
397      *         not of type Transform
398      * @throws IllegalArgumentException if uri is not RFC 2396
399      *         compliant
400      * @throws NullPointerException if dm is null
401      */
402     @SuppressWarnings("rawtypes")
403     public abstract Reference newReference(String uri, DigestMethod dm,
404         List transforms, String type, String id);
405
406     /**
407      * Creates a Reference with the specified parameters and
408      * pre-calculated digest value.
409      *
410      * 

This method is useful when the digest value of a
411      * Reference has been previously computed. See for example,
412      * the
413      * 
414      \* OASIS-DSS \(Digital Signature Services\) specification.
415      *
416      * @param uri the reference URI (may be null)
417      * @param dm the digest method
418      * @param transforms a list of {@link Transform}s. The list is defensively
419      *         copied to protect against subsequent modification. May be
420      *         null or empty.
421      * @param type the reference type, as a URI (may be null)
422      * @param id the reference ID (may be null)


```



```

423     * @param digestValue the digest value. The array is cloned to protect
424     *     against subsequent modification.
425     * @return a <code>Reference</code>
426     * @throws ClassCastException if any of the <code>transforms</code> are
427     *     not of type <code>Transform</code>
428     * @throws IllegalArgumentException if <code>uri</code> is not RFC 2396
429     *     compliant
430     * @throws NullPointerException if <code>dm</code> or
431     *     <code>digestValue</code> is <code>null</code>
432     */
433     @SuppressWarnings("rawtypes")
434     public abstract Reference newReference(String uri, DigestMethod dm,
435         List transforms, String type, String id, byte[] digestValue);
436
437     /**
438     * Creates a <code>Reference</code> with the specified parameters.
439     *
440     * <p>This method is useful when a list of transforms have already been
441     * applied to the <code>Reference</code>. See for example,
442     * the
443     * <a href="http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dss">
444     * OASIS-DSS (Digital Signature Services)</a> specification.
445     *
446     * <p>When an <code>XMLSignature</code> containing this reference is
447     * generated, the specified <code>transforms</code> (if non-null) are
448     * applied to the specified <code>result</code>. The
449     * <code>Transforms</code> element of the resulting <code>Reference</code>
450     * element is set to the concatenation of the
451     * <code>appliedTransforms</code> and <code>transforms</code>.
452     *
453     * @param uri the reference URI (may be <code>null</code>)
454     * @param dm the digest method
455     * @param appliedTransforms a list of {@link Transform}s that have
456     *     already been applied. The list is defensively
457     *     copied to protect against subsequent modification. The list must
458     *     contain at least one entry.
459     * @param result the result of processing the sequence of
460     *     <code>appliedTransforms</code>
461     * @param transforms a list of {@link Transform}s that are to be applied
462     *     when generating the signature. The list is defensively copied to
463     *     protect against subsequent modification. May be <code>null</code>
464     *     or empty.
465     * @param type the reference type, as a URI (may be <code>null</code>)
466     * @param id the reference ID (may be <code>null</code>)
467     * @return a <code>Reference</code>
468     * @throws ClassCastException if any of the transforms (in either list)
469     *     are not of type <code>Transform</code>
470     * @throws IllegalArgumentException if <code>uri</code> is not RFC 2396
471     *     compliant or <code>appliedTransforms</code> is empty
472     * @throws NullPointerException if <code>dm</code>,
473     *     <code>appliedTransforms</code> or <code>result</code> is
474     *     <code>null</code>
475     */

```



```

476 @SuppressWarnings("rawtypes")
477 public abstract Reference newReference(String uri, DigestMethod dm,
478     List appliedTransforms, Data result, List transforms, String type,
479     String id);
480
481 /**
482  * Creates a SignedInfo with the specified canonicalization
483  * and signature methods, and list of one or more references.
484  *
485  * @param cm the canonicalization method
486  * @param sm the signature method
487  * @param references a list of one or more {@link Reference}s. The list is
488  *     defensively copied to protect against subsequent modification.
489  * @return a SignedInfo
490  * @throws ClassCastException if any of the references are not of
491  *     type Reference
492  * @throws IllegalArgumentException if references is empty
493  * @throws NullPointerException if any of the parameters
494  *     are null
495  */
496 @SuppressWarnings("rawtypes")
497 public abstract SignedInfo newSignedInfo(CanonicalizationMethod cm,
498     SignatureMethod sm, List references);
499
500 /**
501  * Creates a SignedInfo with the specified parameters.
502  *
503  * @param cm the canonicalization method
504  * @param sm the signature method
505  * @param references a list of one or more {@link Reference}s. The list is
506  *     defensively copied to protect against subsequent modification.
507  * @param id the id (may be null)
508  * @return a SignedInfo
509  * @throws ClassCastException if any of the references are not of
510  *     type Reference
511  * @throws IllegalArgumentException if references is empty
512  * @throws NullPointerException if cm, sm, or
513  *     references are null
514  */
515 @SuppressWarnings("rawtypes")
516 public abstract SignedInfo newSignedInfo(CanonicalizationMethod cm,
517     SignatureMethod sm, List references, String id);
518
519 // Object factory methods
520 /**
521  * Creates an XMLObject from the specified parameters.
522  *
523  * @param content a list of {@link XMLStructure}s. The list
524  *     is defensively copied to protect against subsequent modification.
525  *     May be null or empty.
526  * @param id the Id (may be null)
527  * @param mimeType the mime type (may be null)
528  * @param encoding the encoding (may be null)

```

```

529     * @return an <code>XMLObject</code>
530     * @throws ClassCastException if <code>content</code> contains any
531     *     entries that are not of type {@link XMLStructure}
532     */
533     @SuppressWarnings("rawtypes")
534     public abstract XMLObject newXMLObject(List content, String id,
535         String mimeType, String encoding);
536
537     /**
538     * Creates a <code>Manifest</code> containing the specified
539     * list of {@link Reference}s.
540     *
541     * @param references a list of one or more <code>Reference</code>s. The list
542     *     is defensively copied to protect against subsequent modification.
543     * @return a <code>Manifest</code>
544     * @throws NullPointerException if <code>references</code> is
545     *     <code>null</code>
546     * @throws IllegalArgumentException if <code>references</code> is empty
547     * @throws ClassCastException if <code>references</code> contains any
548     *     entries that are not of type {@link Reference}
549     */
550     @SuppressWarnings("rawtypes")
551     public abstract Manifest newManifest(List references);
552
553     /**
554     * Creates a <code>Manifest</code> containing the specified
555     * list of {@link Reference}s and optional id.
556     *
557     * @param references a list of one or more <code>Reference</code>s. The list
558     *     is defensively copied to protect against subsequent modification.
559     * @param id the id (may be <code>null</code>)
560     * @return a <code>Manifest</code>
561     * @throws NullPointerException if <code>references</code> is
562     *     <code>null</code>
563     * @throws IllegalArgumentException if <code>references</code> is empty
564     * @throws ClassCastException if <code>references</code> contains any
565     *     entries that are not of type {@link Reference}
566     */
567     @SuppressWarnings("rawtypes")
568     public abstract Manifest newManifest(List references, String id);
569
570     /**
571     * Creates a <code>SignatureProperty</code> containing the specified
572     * list of {@link XMLStructure}s, target URI and optional id.
573     *
574     * @param content a list of one or more <code>XMLStructure</code>s. The list
575     *     is defensively copied to protect against subsequent modification.
576     * @param target the target URI of the Signature that this property applies
577     *     to
578     * @param id the id (may be <code>null</code>)
579     * @return a <code>SignatureProperty</code>
580     * @throws NullPointerException if <code>content</code> or
581     *     <code>target</code> is <code>null</code>

```

```

582     * @throws IllegalArgumentException if <code>content</code> is empty
583     * @throws ClassCastException if <code>content</code> contains any
584     *     entries that are not of type {@link XMLStructure}
585     */
586     @SuppressWarnings("rawtypes")
587     public abstract SignatureProperty newSignatureProperty
588         (List content, String target, String id);
589
590     /**
591     * Creates a <code>SignatureProperties</code> containing the specified
592     * list of {@link SignatureProperty}s and optional id.
593     *
594     * @param properties a list of one or more <code>SignatureProperty</code>s.
595     *     The list is defensively copied to protect against subsequent
596     *     modification.
597     * @param id the id (may be <code>null</code>)
598     * @return a <code>SignatureProperties</code>
599     * @throws NullPointerException if <code>properties</code>
600     *     is <code>null</code>
601     * @throws IllegalArgumentException if <code>properties</code> is empty
602     * @throws ClassCastException if <code>properties</code> contains any
603     *     entries that are not of type {@link SignatureProperty}
604     */
605     @SuppressWarnings("rawtypes")
606     public abstract SignatureProperties newSignatureProperties
607         (List properties, String id);
608
609     // Algorithm factory methods
610     /**
611     * Creates a <code>DigestMethod</code> for the specified algorithm URI
612     * and parameters.
613     *
614     * @param algorithm the URI identifying the digest algorithm
615     * @param params algorithm-specific digest parameters (may be
616     *     <code>null</code>)
617     * @return the <code>DigestMethod</code>
618     * @throws InvalidAlgorithmParameterException if the specified parameters
619     *     are inappropriate for the requested algorithm
620     * @throws NoSuchAlgorithmException if an implementation of the
621     *     specified algorithm cannot be found
622     * @throws NullPointerException if <code>algorithm</code> is
623     *     <code>null</code>
624     */
625     public abstract DigestMethod newDigestMethod(String algorithm,
626         DigestMethodParameterSpec params) throws NoSuchAlgorithmException,
627         InvalidAlgorithmParameterException;
628
629     /**
630     * Creates a <code>SignatureMethod</code> for the specified algorithm URI
631     * and parameters.
632     *
633     * @param algorithm the URI identifying the signature algorithm
634     * @param params algorithm-specific signature parameters (may be

```

```

635     * <code>null</code>))
636     * @return the <code>SignatureMethod</code>
637     * @throws InvalidAlgorithmParameterException if the specified parameters
638     *     are inappropriate for the requested algorithm
639     * @throws NoSuchAlgorithmException if an implementation of the
640     *     specified algorithm cannot be found
641     * @throws NullPointerException if <code>algorithm</code> is
642     *     <code>null</code>
643     */
644     public abstract SignatureMethod newSignatureMethod(String algorithm,
645         SignatureMethodParameterSpec params) throws NoSuchAlgorithmException,
646         InvalidAlgorithmParameterException;
647
648     /**
649     * Creates a <code>Transform</code> for the specified algorithm URI
650     * and parameters.
651     *
652     * @param algorithm the URI identifying the transform algorithm
653     * @param params algorithm-specific transform parameters (may be
654     *     <code>null</code>)
655     * @return the <code>Transform</code>
656     * @throws InvalidAlgorithmParameterException if the specified parameters
657     *     are inappropriate for the requested algorithm
658     * @throws NoSuchAlgorithmException if an implementation of the
659     *     specified algorithm cannot be found
660     * @throws NullPointerException if <code>algorithm</code> is
661     *     <code>null</code>
662     */
663     public abstract Transform newTransform(String algorithm,
664         TransformParameterSpec params) throws NoSuchAlgorithmException,
665         InvalidAlgorithmParameterException;
666
667     /**
668     * Creates a <code>Transform</code> for the specified algorithm URI
669     * and parameters. The parameters are specified as a mechanism-specific
670     * <code>XMLStructure</code> (ex: {@link DOMStructure}). This method is
671     * useful when the parameters are in XML form or there is no standard
672     * class for specifying the parameters.
673     *
674     * @param algorithm the URI identifying the transform algorithm
675     * @param params a mechanism-specific XML structure from which to
676     *     unmarshal the parameters from (may be <code>null</code> if
677     *     not required or optional)
678     * @return the <code>Transform</code>
679     * @throws ClassCastException if the type of <code>params</code> is
680     *     inappropriate for this <code>XMLSignatureFactory</code>
681     * @throws InvalidAlgorithmParameterException if the specified parameters
682     *     are inappropriate for the requested algorithm
683     * @throws NoSuchAlgorithmException if an implementation of the
684     *     specified algorithm cannot be found
685     * @throws NullPointerException if <code>algorithm</code> is
686     *     <code>null</code>
687     */

```

```

688 public abstract Transform newTransform(String algorithm,
689     XMLStructure params) throws NoSuchAlgorithmException,
690     InvalidAlgorithmParameterException;
691
692 /**
693  * Creates a CanonicalizationMethod for the specified
694  * algorithm URI and parameters.
695  *
696  * @param algorithm the URI identifying the canonicalization algorithm
697  * @param params algorithm-specific canonicalization parameters (may be
698  *     null)
699  * @return the CanonicalizationMethod
700  * @throws InvalidAlgorithmParameterException if the specified parameters
701  *     are inappropriate for the requested algorithm
702  * @throws NoSuchAlgorithmException if an implementation of the
703  *     specified algorithm cannot be found
704  * @throws NullPointerException if algorithm is
705  *     null
706  */
707 public abstract CanonicalizationMethod newCanonicalizationMethod(
708     String algorithm, C14NMethodParameterSpec params)
709     throws NoSuchAlgorithmException, InvalidAlgorithmParameterException;
710
711 /**
712  * Creates a CanonicalizationMethod for the specified
713  * algorithm URI and parameters. The parameters are specified as a
714  * mechanism-specific XMLStructure (ex: {@link DOMStructure}).
715  * This method is useful when the parameters are in XML form or there is
716  * no standard class for specifying the parameters.
717  *
718  * @param algorithm the URI identifying the canonicalization algorithm
719  * @param params a mechanism-specific XML structure from which to
720  *     unmarshal the parameters from (may be null if
721  *     not required or optional)
722  * @return the CanonicalizationMethod
723  * @throws ClassCastException if the type of params is
724  *     inappropriate for this XMLSignatureFactory
725  * @throws InvalidAlgorithmParameterException if the specified parameters
726  *     are inappropriate for the requested algorithm
727  * @throws NoSuchAlgorithmException if an implementation of the
728  *     specified algorithm cannot be found
729  * @throws NullPointerException if algorithm is
730  *     null
731  */
732 public abstract CanonicalizationMethod newCanonicalizationMethod(
733     String algorithm, XMLStructure params)
734     throws NoSuchAlgorithmException, InvalidAlgorithmParameterException;
735
736 /**
737  * Returns a KeyInfoFactory that creates KeyInfo
738  * objects. The returned KeyInfoFactory has the same
739  * mechanism type and provider as this XMLSignatureFactory.
740  *

```

```

741     * @return a <code>KeyInfoFactory</code>
742     * @throws NoSuchMechanismException if a <code>KeyFactory</code>
743     *     implementation with the same mechanism type and provider
744     *     is not available
745     */
746     public final KeyInfoFactory getKeyInfoFactory() {
747         return KeyInfoFactory.getInstance(getMechanismType(), getProvider());
748     }
749
750     /**
751     * Unmarshals a new <code>XMLSignature</code> instance from a
752     * mechanism-specific <code>XMLValidateContext</code> instance.
753     *
754     * @param context a mechanism-specific context from which to unmarshal the
755     *     signature from
756     * @return the <code>XMLSignature</code>
757     * @throws NullPointerException if <code>context</code> is
758     *     <code>null</code>
759     * @throws ClassCastException if the type of <code>context</code> is
760     *     inappropriate for this factory
761     * @throws MarshalException if an unrecoverable exception occurs
762     *     during unmarshalling
763     */
764     public abstract XMLSignature unmarshalXMLSignature
765         (XMLValidateContext context) throws MarshalException;
766
767     /**
768     * Unmarshals a new <code>XMLSignature</code> instance from a
769     * mechanism-specific <code>XMLStructure</code> instance.
770     * This method is useful if you only want to unmarshal (and not
771     * validate) an <code>XMLSignature</code>.
772     *
773     * @param xmlStructure a mechanism-specific XML structure from which to
774     *     unmarshal the signature from
775     * @return the <code>XMLSignature</code>
776     * @throws NullPointerException if <code>xmlStructure</code> is
777     *     <code>null</code>
778     * @throws ClassCastException if the type of <code>xmlStructure</code> is
779     *     inappropriate for this factory
780     * @throws MarshalException if an unrecoverable exception occurs
781     *     during unmarshalling
782     */
783     public abstract XMLSignature unmarshalXMLSignature
784         (XMLStructure xmlStructure) throws MarshalException;
785
786     /**
787     * Indicates whether a specified feature is supported.
788     *
789     * @param feature the feature name (as an absolute URI)
790     * @return <code>true</code> if the specified feature is supported,
791     *     <code>false</code> otherwise
792     * @throws NullPointerException if <code>feature</code> is <code>null</code>
793     */

```

```

794     public abstract boolean isFeatureSupported(String feature);
795
796     /**
797      * Returns a reference to the URIDereferencer that is used by
798      * default to dereference URIs in {@link Reference} objects.
799      *
800      * @return a reference to the default URIDereferencer (never
801      *         null)
802      */
803     public abstract URIDereferencer getURIDereferencer();
804 }

```

## 10.17 XMLValidateContext.java

Secuencia 124: javax/xml/crypto/dsig/XMLValidateContext.java

```

1  /*
2   * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25 /*
26  * $Id: XMLValidateContext.java,v 1.8 2005/05/10 16:03:49 mullan Exp $
27  */
28 package javax.xml.crypto.dsig;
29
30 import javax.xml.crypto.XMLCryptoContext;
31
32 /**
33  * Contains context information for validating XML Signatures. This interface
34  * is primarily intended for type-safety.
35  *
36  * <p>Note that XMLValidateContext instances can contain
37  * information and state specific to the XML signature structure it is
38  * used with. The results are unpredictable if an

```



```

39 * <code>XMLValidateContext</code> is used with different signature structures
40 * (for example, you should not use the same <code>XMLValidateContext</code>
41 * instance to validate two different {@link XMLSignature} objects).
42 * <p>
43 * <b><a name="Supported Properties"></a>Supported Properties</b>
44 * <p>The following properties can be set by an application using the
45 * {@link #setProperty setProperty} method.
46 * <ul>
47 * <li><code>javax.xml.crypto.dsig.cacheReference</code>: value must be a
48 *   {@link Boolean}. This property controls whether or not the
49 *   {@link Reference#validate Reference.validate} method will cache the
50 *   dereferenced content and pre-digested input for subsequent retrieval via
51 *   the {@link Reference#getDereferencedData Reference.getDereferencedData}
52 *   and {@link Reference#getDigestInputStream
53 *   Reference.getDigestInputStream} methods. The default value if not
54 *   specified is <code>Boolean.FALSE</code>.
55 * </ul>
56 *
57 * @author Sean Mullan
58 * @author JSR 105 Expert Group
59 * @since 1.6
60 * @see XMLSignature#validate(XMLValidateContext)
61 * @see Reference#validate(XMLValidateContext)
62 */
63 public interface XMLValidateContext extends XMLCryptoContext {}

```

## 11 Xml Crypto Dsig Dom (javax.xml.crypto.dsig.dom package)

### 11.1 DOMSignContext.java

Secuencia 125: javax/xml/crypto/dsig/dom/DOMSignContext.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *

```



```

23  *
24  */
25  /*
26  * $Id: DOMSignContext.java,v 1.9 2005/05/10 16:31:14 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.dom;
29
30  import javax.xml.crypto.KeySelector;
31  import javax.xml.crypto.dom.DOMCryptoContext;
32  import javax.xml.crypto.dsig.XMLSignContext;
33  import javax.xml.crypto.dsig.XMLSignature;
34  import java.security.Key;
35  import org.w3c.dom.Node;
36
37  /**
38   * A DOM-specific {@link XMLSignContext}. This class contains additional methods
39   * to specify the location in a DOM tree where an {@link XMLSignature}
40   * object is to be marshalled when generating the signature.
41   *
42   * <p>Note that <code>DOMSignContext</code> instances can contain
43   * information and state specific to the XML signature structure it is
44   * used with. The results are unpredictable if a
45   * <code>DOMSignContext</code> is used with different signature structures
46   * (for example, you should not use the same <code>DOMSignContext</code>
47   * instance to sign two different {@link XMLSignature} objects).
48   *
49   * @author Sean Mullan
50   * @author JSR 105 Expert Group
51   * @since 1.6
52   */
53  public class DOMSignContext extends DOMCryptoContext implements XMLSignContext {
54
55      private Node parent;
56      private Node nextSibling;
57
58      /**
59       * Creates a <code>DOMSignContext</code> with the specified signing key
60       * and parent node. The signing key is stored in a
61       * {@link KeySelector#singletonKeySelector singleton KeySelector} that is
62       * returned by the {@link #getKeySelector getKeySelector} method.
63       * The marshalled <code>XMLSignature</code> will be added as the last
64       * child element of the specified parent node unless a next sibling node is
65       * specified by invoking the {@link #setNextSibling setNextSibling} method.
66       *
67       * @param signingKey the signing key
68       * @param parent the parent node
69       * @throws NullPointerException if <code>signingKey</code> or
70       *         <code>parent</code> is <code>null</code>
71       */
72      public DOMSignContext(Key signingKey, Node parent) {
73          if (signingKey == null) {
74              throw new NullPointerException("signingKey cannot be null");
75          }
76      }
77  }

```

```

76     if (parent == null) {
77         throw new NullPointerException("parent cannot be null");
78     }
79     setKeySelector(KeySelector.singletonKeySelector(signingKey));
80     this.parent = parent;
81 }
82
83 /**
84  * Creates a DOMSignContext with the specified signing key,
85  * parent and next sibling nodes. The signing key is stored in a
86  * {@link KeySelector#singletonKeySelector singleton KeySelector} that is
87  * returned by the {@link #getKeySelector getKeySelector} method.
88  * The marshalled XMLSignature will be inserted as a child
89  * element of the specified parent node and immediately before the
90  * specified next sibling node.
91  *
92  * @param signingKey the signing key
93  * @param parent the parent node
94  * @param nextSibling the next sibling node
95  * @throws NullPointerException if signingKey,
96  *     parent or nextSibling is null
97  */
98 public DOMSignContext(Key signingKey, Node parent, Node nextSibling) {
99     if (signingKey == null) {
100         throw new NullPointerException("signingKey cannot be null");
101     }
102     if (parent == null) {
103         throw new NullPointerException("parent cannot be null");
104     }
105     if (nextSibling == null) {
106         throw new NullPointerException("nextSibling cannot be null");
107     }
108     setKeySelector(KeySelector.singletonKeySelector(signingKey));
109     this.parent = parent;
110     this.nextSibling = nextSibling;
111 }
112
113 /**
114  * Creates a DOMSignContext with the specified key selector
115  * and parent node. The marshalled XMLSignature will be added
116  * as the last child element of the specified parent node unless a next
117  * sibling node is specified by invoking the
118  * {@link #setNextSibling setNextSibling} method.
119  *
120  * @param ks the key selector
121  * @param parent the parent node
122  * @throws NullPointerException if ks or parent
123  *     is null
124  */
125 public DOMSignContext(KeySelector ks, Node parent) {
126     if (ks == null) {
127         throw new NullPointerException("key selector cannot be null");
128     }

```

```
129     if (parent == null) {
130         throw new NullPointerException("parent cannot be null");
131     }
132     setKeySelector(ks);
133     this.parent = parent;
134 }
135
136 /**
137  * Creates a DOMSignContext with the specified key selector,
138  * parent and next sibling nodes. The marshalled XMLSignature
139  * will be inserted as a child element of the specified parent node and
140  * immediately before the specified next sibling node.
141  *
142  * @param ks the key selector
143  * @param parent the parent node
144  * @param nextSibling the next sibling node
145  * @throws NullPointerException if ks, parent or
146  *         nextSibling is null
147  */
148 public DOMSignContext(KeySelector ks, Node parent, Node nextSibling) {
149     if (ks == null) {
150         throw new NullPointerException("key selector cannot be null");
151     }
152     if (parent == null) {
153         throw new NullPointerException("parent cannot be null");
154     }
155     if (nextSibling == null) {
156         throw new NullPointerException("nextSibling cannot be null");
157     }
158     setKeySelector(ks);
159     this.parent = parent;
160     this.nextSibling = nextSibling;
161 }
162
163 /**
164  * Sets the parent node.
165  *
166  * @param parent the parent node. The marshalled XMLSignature
167  * will be added as a child element of this node.
168  * @throws NullPointerException if parent is null
169  * @see #getParent
170  */
171 public void setParent(Node parent) {
172     if (parent == null) {
173         throw new NullPointerException("parent is null");
174     }
175     this.parent = parent;
176 }
177
178 /**
179  * Sets the next sibling node.
180  *
181  * @param nextSibling the next sibling node. The marshalled
```

```

182     * <code>XMLSignature</code> will be inserted immediately before this
183     * node. Specify <code>null</code> to remove the current setting.
184     * @see #getNextSibling
185     */
186     public void setNextSibling(Node nextSibling) {
187         this.nextSibling = nextSibling;
188     }
189
190     /**
191     * Returns the parent node.
192     *
193     * @return the parent node (never <code>null</code>)
194     * @see #setParent(Node)
195     */
196     public Node getParent() {
197         return parent;
198     }
199
200     /**
201     * Returns the nextSibling node.
202     *
203     * @return the nextSibling node, or <code>null</code> if not specified.
204     * @see #setNextSibling(Node)
205     */
206     public Node getNextSibling() {
207         return nextSibling;
208     }
209 }

```

## 11.2 DOMValidateContext.java

Secuencia 126: javax/xml/crypto/dsig/dom/DOMValidateContext.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```

22  *
23  *
24  */
25  /*
26  * $Id: DOMValidateContext.java,v 1.8 2005/05/10 16:31:14 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.dom;
29
30  import javax.xml.crypto.KeySelector;
31  import javax.xml.crypto.dom.DOMCryptoContext;
32  import javax.xml.crypto.dsig.XMLSignature;
33  import javax.xml.crypto.dsig.XMLSignatureFactory;
34  import javax.xml.crypto.dsig.XMLValidateContext;
35  import java.security.Key;
36  import org.w3c.dom.Node;
37
38  /**
39   * A DOM-specific {@link XMLValidateContext}. This class contains additional
40   * methods to specify the location in a DOM tree where an {@link XMLSignature}
41   * is to be unmarshalled and validated from.
42   *
43   * <p>Note that the behavior of an unmarshalled <code>XMLSignature</code>
44   * is undefined if the contents of the underlying DOM tree are modified by the
45   * caller after the <code>XMLSignature</code> is created.
46   *
47   * <p>Also, note that <code>DOMValidateContext</code> instances can contain
48   * information and state specific to the XML signature structure it is
49   * used with. The results are unpredictable if a
50   * <code>DOMValidateContext</code> is used with different signature structures
51   * (for example, you should not use the same <code>DOMValidateContext</code>
52   * instance to validate two different {@link XMLSignature} objects).
53   *
54   * @author Sean Mullan
55   * @author JSR 105 Expert Group
56   * @since 1.6
57   * @see XMLSignatureFactory#unmarshalXMLSignature(XMLValidateContext)
58   */
59  public class DOMValidateContext extends DOMCryptoContext
60      implements XMLValidateContext {
61
62      private Node node;
63
64      /**
65       * Creates a <code>DOMValidateContext</code> containing the specified key
66       * selector and node.
67       *
68       * @param ks a key selector for finding a validation key
69       * @param node the node
70       * @throws NullPointerException if <code>ks</code> or <code>node</code> is
71       *         <code>null</code>
72       */
73      public DOMValidateContext(KeySelector ks, Node node) {
74          if (ks == null) {

```

```

75         throw new NullPointerException("key selector is null");
76     }
77     init(node, ks);
78 }
79
80 /**
81  * Creates a <code>DOMValidateContext</code> containing the specified key
82  * and node. The validating key will be stored in a
83  * {@link KeySelector#singletonKeySelector singleton KeySelector} that
84  * is returned when the {@link #getKeySelector getKeySelector}
85  * method is called.
86  *
87  * @param validatingKey the validating key
88  * @param node the node
89  * @throws NullPointerException if <code>validatingKey</code> or
90  *     <code>node</code> is <code>null</code>
91  */
92 public DOMValidateContext(Key validatingKey, Node node) {
93     if (validatingKey == null) {
94         throw new NullPointerException("validatingKey is null");
95     }
96     init(node, KeySelector.singletonKeySelector(validatingKey));
97 }
98
99 private void init(Node node, KeySelector ks) {
100     if (node == null) {
101         throw new NullPointerException("node is null");
102     }
103
104     this.node = node;
105     super.setKeySelector(ks);
106     if (System.getSecurityManager() != null) {
107         super.setProperty("org.jcp.xml.dsig.secureValidation",
108             Boolean.TRUE);
109     }
110 }
111
112 /**
113  * Sets the node.
114  *
115  * @param node the node
116  * @throws NullPointerException if <code>node</code> is <code>null</code>
117  * @see #getNode
118  */
119 public void setNode(Node node) {
120     if (node == null) {
121         throw new NullPointerException();
122     }
123     this.node = node;
124 }
125
126 /**
127  * Returns the node.

```

```

128     *
129     * @return the node (never <code>null</code>)
130     * @see #setNode(Node)
131     */
132     public Node getNode() {
133         return node;
134     }
135 }

```

## 12 Xml Crypto Dsig Keyinfo (javax.xml.crypto.dsig.keyinfo package)

### 12.1 KeyInfo.java

Secuencia 127: javax/xml/crypto/dsig/keyinfo/KeyInfo.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: KeyInfo.java,v 1.7 2005/05/10 16:35:34 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.keyinfo;
29
30 import java.util.List;
31 import javax.xml.crypto.MarshalException;
32 import javax.xml.crypto.XMLCryptoContext;
33 import javax.xml.crypto.XMLStructure;
34
35 /**
36 * A representation of the XML <code>KeyInfo</code> element as defined in
37 * the <a href="http://www.w3.org/TR/xmlldsig-core/">

```

```

38 * W3C Recommendation for XML-Signature Syntax and Processing</a>.
39 * A <code>KeyInfo</code> contains a list of {@link XMLStructure}s, each of
40 * which contain information that enables the recipient(s) to obtain the key
41 * needed to validate an XML signature. The XML Schema Definition is defined as:
42 *
43 * <pre>
44 * <lt;element name="KeyInfo" type="ds:KeyInfoType"/>
45 * <lt;complexType name="KeyInfoType" mixed="true">
46 *   <lt;choice maxOccurs="unbounded">
47 *     <lt;element ref="ds:KeyName"/>
48 *     <lt;element ref="ds:KeyValue"/>
49 *     <lt;element ref="ds:RetrievalMethod"/>
50 *     <lt;element ref="ds:X509Data"/>
51 *     <lt;element ref="ds:PGPData"/>
52 *     <lt;element ref="ds:SPKIData"/>
53 *     <lt;element ref="ds:MgmtData"/>
54 *     <lt;any processContents="lax" namespace="##other"/>
55 *     <lt;!-- (1,1) elements from (0,unbounded) namespaces -->
56 *   <lt;/choice>
57 *   <lt;attribute name="Id" type="ID" use="optional"/>
58 * <lt;/complexType>
59 * </pre>
60 *
61 * A <code>KeyInfo</code> instance may be created by invoking one of the
62 * {@link KeyInfoFactory#newKeyInfo newKeyInfo} methods of the
63 * {@link KeyInfoFactory} class, and passing it a list of one or more
64 * <code>XMLStructure</code>s and an optional id parameter;
65 * for example:
66 * <pre>
67 *   KeyInfoFactory factory = KeyInfoFactory.getInstance("DOM");
68 *   KeyInfo keyInfo = factory.newKeyInfo
69 *     (Collections.singletonList(factory.newKeyName("Alice"), "keyinfo-1"));
70 * </pre>
71 *
72 * <p><code>KeyInfo</code> objects can also be marshalled to XML by invoking
73 * the {@link #marshal marshal} method.
74 *
75 * @author Sean Mullan
76 * @author JSR 105 Expert Group
77 * @since 1.6
78 * @see KeyInfoFactory#newKeyInfo(List)
79 * @see KeyInfoFactory#newKeyInfo(List, String)
80 */
81 public interface KeyInfo extends XMLStructure {
82
83   /**
84    * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
85    * list} containing the key information. Each entry of the list is
86    * an {@link XMLStructure}.
87    *
88    * <p>If there is a public subclass representing the type of
89    * <code>XMLStructure</code>, it is returned as an instance of that
90    * class (ex: an <code>X509Data</code> element would be returned as an

```



```

91     * instance of {@link javax.xml.crypto.dsig.keyinfo.X509Data}).
92     *
93     * @return an unmodifiable list of one or more <code>XMLStructure</code>s
94     *     in this <code>KeyInfo</code>. Never returns <code>null</code> or an
95     *     empty list.
96     */
97     @SuppressWarnings("rawtypes")
98     List getContent();
99
100    /**
101     * Return the optional Id attribute of this <code>KeyInfo</code>, which
102     * may be useful for referencing this <code>KeyInfo</code> from other
103     * XML structures.
104     *
105     * @return the Id attribute of this <code>KeyInfo</code> (may be
106     *     <code>null</code> if not specified)
107     */
108     String getId();
109
110    /**
111     * Marshals the key info to XML.
112     *
113     * @param parent a mechanism-specific structure containing the parent node
114     *     that the marshalled key info will be appended to
115     * @param context the <code>XMLCryptoContext</code> containing additional
116     *     context (may be null if not applicable)
117     * @throws ClassCastException if the type of <code>parent</code> or
118     *     <code>context</code> is not compatible with this key info
119     * @throws MarshalException if the key info cannot be marshalled
120     * @throws NullPointerException if <code>parent</code> is <code>null</code>
121     */
122     void marshal(XMLStructure parent, XMLCryptoContext context)
123         throws MarshalException;
124 }

```

## 12.2 KeyInfoFactory.java

Secuencia 128: javax/xml/crypto/dsig/keyinfo/KeyInfoFactory.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: KeyInfoFactory.java,v 1.12 2005/05/10 16:35:35 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.keyinfo;
29
30  import java.math.BigInteger;
31  import java.security.KeyException;
32  import java.security.NoSuchAlgorithmException;
33  import java.security.NoSuchProviderException;
34  import java.security.Provider;
35  import java.security.PublicKey;
36  import java.security.Security;
37  import java.security.cert.X509CRL;
38  import java.util.List;
39  import javax.xml.crypto.MarshalException;
40  import javax.xml.crypto.NoSuchMechanismException;
41  import javax.xml.crypto.URIDereferencer;
42  import javax.xml.crypto.XMLStructure;
43  import javax.xml.crypto.dom.DOMStructure;
44  import javax.xml.crypto.dsig.*;
45
46  import sun.security.jca.*;
47  import sun.security.jca.GetInstance.Instance;
48
49  /**
50   * A factory for creating {@link KeyInfo} objects from scratch or for
51   * unmarshalling a <code>KeyInfo</code> object from a corresponding XML
52   * representation.
53   *
54   * <p>Each instance of <code>KeyInfoFactory</code> supports a specific
55   * XML mechanism type. To create a <code>KeyInfoFactory</code>, call one of the
56   * static {@link #getInstance getInstance} methods, passing in the XML
57   * mechanism type desired, for example:
58   *
59   * <blockquote><code>
60   *     KeyInfoFactory factory = KeyInfoFactory.getInstance("DOM");
61   * </code></blockquote>
62   *
63   * <p>The objects that this factory produces will be based
64   * on DOM and abide by the DOM interoperability requirements as defined in the
65   * <a href="../../../../technotes/guides/security/xmlldsig/overview.html#DOM Mechanism
66   *     Requirements">
67   * DOM Mechanism Requirements</a> section of the API overview. See the
68   * <a href="../../../../technotes/guides/security/xmlldsig/overview.html#Service Provider">

```

```

68  * Service Providers</a> section of the API overview for a list of standard
69  * mechanism types.
70  *
71  * <p><code>KeyInfoFactory</code> implementations are registered and loaded
72  * using the {@link java.security.Provider} mechanism.
73  * For example, a service provider that supports the
74  * DOM mechanism would be specified in the <code>Provider</code> subclass as:
75  * <pre>
76  *     put("KeyInfoFactory.DOM", "org.example.DOMKeyInfoFactory");
77  * </pre>
78  *
79  * <p>Also, the <code>XMLStructure</code>s that are created by this factory
80  * may contain state specific to the <code>KeyInfo</code> and are not
81  * intended to be reusable.
82  *
83  * <p>An implementation MUST minimally support the default mechanism type: DOM.
84  *
85  * <p>Note that a caller must use the same <code>KeyInfoFactory</code>
86  * instance to create the <code>XMLStructure</code>s of a particular
87  * <code>KeyInfo</code> object. The behavior is undefined if
88  * <code>XMLStructure</code>s from different providers or different mechanism
89  * types are used together.
90  *
91  * <p><b>Concurrent Access</b>
92  * <p>The static methods of this class are guaranteed to be thread-safe.
93  * Multiple threads may concurrently invoke the static methods defined in this
94  * class with no ill effects.
95  *
96  * <p>However, this is not true for the non-static methods defined by this
97  * class. Unless otherwise documented by a specific provider, threads that
98  * need to access a single <code>KeyInfoFactory</code> instance concurrently
99  * should synchronize amongst themselves and provide the necessary locking.
100  * Multiple threads each manipulating a different <code>KeyInfoFactory</code>
101  * instance need not synchronize.
102  *
103  * @author Sean Mullan
104  * @author JSR 105 Expert Group
105  * @since 1.6
106  */
107 public abstract class KeyInfoFactory {
108
109     private String mechanismType;
110     private Provider provider;
111
112     /**
113      * Default constructor, for invocation by subclasses.
114      */
115     protected KeyInfoFactory() {}
116
117     /**
118      * Returns a <code>KeyInfoFactory</code> that supports the
119      * specified XML processing mechanism and representation type (ex: "DOM").
120      *

```

```

121 * <p>This method uses the standard JCA provider lookup mechanism to
122 * locate and instantiate a <code>KeyInfoFactory</code> implementation of
123 * the desired mechanism type. It traverses the list of registered security
124 * <code>Provider</code>s, starting with the most preferred
125 * <code>Provider</code>. A new <code>KeyInfoFactory</code> object
126 * from the first <code>Provider</code> that supports the specified
127 * mechanism is returned.
128 *
129 * <p> Note that the list of registered providers may be retrieved via
130 * the {@link Security#getProviders() Security.getProviders()} method.
131 *
132 * @param mechanismType the type of the XML processing mechanism and
133 * representation. See the <a
134 * href="../../../../../../technotes/guides/security/xmlsig/overview.html#Service Provider
135 * Service Providers</a> section of the API overview for a list of
136 * standard mechanism types.
137 * @return a new <code>KeyInfoFactory</code>
138 * @throws NullPointerException if <code>mechanismType</code> is
139 * <code>null</code>
140 * @throws NoSuchMechanismException if no <code>Provider</code> supports a
141 * <code>KeyInfoFactory</code> implementation for the specified mechanism
142 * @see Provider
143 */
144 public static KeyInfoFactory getInstance(String mechanismType) {
145     if (mechanismType == null) {
146         throw new NullPointerException("mechanismType cannot be null");
147     }
148     Instance instance;
149     try {
150         instance = GetInstance.getInstance
151             ("KeyInfoFactory", null, mechanismType);
152     } catch (NoSuchAlgorithmException nsae) {
153         throw new NoSuchMechanismException(nsae);
154     }
155     KeyInfoFactory factory = (KeyInfoFactory) instance.impl;
156     factory.mechanismType = mechanismType;
157     factory.provider = instance.provider;
158     return factory;
159 }
160
161 /**
162 * Returns a <code>KeyInfoFactory</code> that supports the
163 * requested XML processing mechanism and representation type (ex: "DOM"),
164 * as supplied by the specified provider. Note that the specified
165 * <code>Provider</code> object does not have to be registered in the
166 * provider list.
167 *
168 * @param mechanismType the type of the XML processing mechanism and
169 * representation. See the <a
170 * href="../../../../../../technotes/guides/security/xmlsig/overview.html#Service Provider
171 * Service Providers</a> section of the API overview for a list of

```

```

172     * standard mechanism types.
173     * @param provider the <code>Provider</code> object
174     * @return a new <code>KeyInfoFactory</code>
175     * @throws NullPointerException if <code>mechanismType</code> or
176     * <code>provider</code> are <code>null</code>
177     * @throws NoSuchMechanismException if a <code>KeyInfoFactory</code>
178     * implementation for the specified mechanism is not available from the
179     * specified <code>Provider</code> object
180     * @see Provider
181     */
182     public static KeyInfoFactory getInstance(String mechanismType,
183         Provider provider) {
184         if (mechanismType == null) {
185             throw new NullPointerException("mechanismType cannot be null");
186         } else if (provider == null) {
187             throw new NullPointerException("provider cannot be null");
188         }
189
190         Instance instance;
191         try {
192             instance = GetInstance.getInstance
193                 ("KeyInfoFactory", null, mechanismType, provider);
194         } catch (NoSuchAlgorithmException nsae) {
195             throw new NoSuchMechanismException(nsae);
196         }
197         KeyInfoFactory factory = (KeyInfoFactory) instance.impl;
198         factory.mechanismType = mechanismType;
199         factory.provider = instance.provider;
200         return factory;
201     }
202
203     /**
204     * Returns a <code>KeyInfoFactory</code> that supports the
205     * requested XML processing mechanism and representation type (ex: "DOM"),
206     * as supplied by the specified provider. The specified provider must be
207     * registered in the security provider list.
208     *
209     * <p>Note that the list of registered providers may be retrieved via
210     * the {@link Security#getProviders() Security.getProviders()} method.
211     *
212     * @param mechanismType the type of the XML processing mechanism and
213     * representation. See the <a
214     * href="http://www.oracle.com/technetes/guides/security/xmlsig/overview.html#Service Provider
215     * ">
216     * Service Providers</a> section of the API overview for a list of
217     * standard mechanism types.
218     * @param provider the string name of the provider
219     * @return a new <code>KeyInfoFactory</code>
220     * @throws NoSuchProviderException if the specified provider is not
221     * registered in the security provider list
222     * @throws NullPointerException if <code>mechanismType</code> or
223     * <code>provider</code> are <code>null</code>
224     * @throws NoSuchMechanismException if a <code>KeyInfoFactory</code>

```

```

224     * implementation for the specified mechanism is not available from the
225     * specified provider
226     * @see Provider
227     */
228     public static KeyInfoFactory getInstance(String mechanismType,
229     String provider) throws NoSuchProviderException {
230         if (mechanismType == null) {
231             throw new NullPointerException("mechanismType cannot be null");
232         } else if (provider == null) {
233             throw new NullPointerException("provider cannot be null");
234         } else if (provider.length() == 0) {
235             throw new NoSuchProviderException();
236         }
237
238         Instance instance;
239         try {
240             instance = GetInstance.getInstance
241                 ("KeyInfoFactory", null, mechanismType, provider);
242         } catch (NoSuchAlgorithmException nsae) {
243             throw new NoSuchMechanismException(nsae);
244         }
245         KeyInfoFactory factory = (KeyInfoFactory) instance.impl;
246         factory.mechanismType = mechanismType;
247         factory.provider = instance.provider;
248         return factory;
249     }
250
251     /**
252     * Returns a <code>KeyInfoFactory</code> that supports the
253     * default XML processing mechanism and representation type ("DOM").
254     *
255     * <p>This method uses the standard JCA provider lookup mechanism to
256     * locate and instantiate a <code>KeyInfoFactory</code> implementation of
257     * the default mechanism type. It traverses the list of registered security
258     * <code>Provider</code>s, starting with the most preferred
259     * <code>Provider</code>. A new <code>KeyInfoFactory</code> object
260     * from the first <code>Provider</code> that supports the DOM mechanism is
261     * returned.
262     *
263     * <p> Note that the list of registered providers may be retrieved via
264     * the {@link Security#getProviders() Security.getProviders()} method.
265     *
266     * @return a new <code>KeyInfoFactory</code>
267     * @throws NoSuchMechanismException if no <code>Provider</code> supports a
268     * <code>KeyInfoFactory</code> implementation for the DOM mechanism
269     * @see Provider
270     */
271     public static KeyInfoFactory getInstance() {
272         return getInstance("DOM");
273     }
274
275     /**
276     * Returns the type of the XML processing mechanism and representation

```

```

277     * supported by this <code>KeyInfoFactory</code> (ex: "DOM")
278     *
279     * @return the XML processing mechanism type supported by this
280     *     <code>KeyInfoFactory</code>
281     */
282     public final String getMechanismType() {
283         return mechanismType;
284     }
285
286     /**
287     * Returns the provider of this <code>KeyInfoFactory</code>.
288     *
289     * @return the provider of this <code>KeyInfoFactory</code>
290     */
291     public final Provider getProvider() {
292         return provider;
293     }
294
295     /**
296     * Creates a <code>KeyInfo</code> containing the specified list of
297     * key information types.
298     *
299     * @param content a list of one or more {@link XMLStructure}s representing
300     *     key information types. The list is defensively copied to protect
301     *     against subsequent modification.
302     * @return a <code>KeyInfo</code>
303     * @throws NullPointerException if <code>content</code> is <code>>null</code>
304     * @throws IllegalArgumentException if <code>content</code> is empty
305     * @throws ClassCastException if <code>content</code> contains any entries
306     *     that are not of type {@link XMLStructure}
307     */
308     @SuppressWarnings("rawtypes")
309     public abstract KeyInfo newKeyInfo(List content);
310
311     /**
312     * Creates a <code>KeyInfo</code> containing the specified list of key
313     * information types and optional id. The
314     * <code>id</code> parameter represents the value of an XML
315     * <code>ID</code> attribute and is useful for referencing
316     * the <code>KeyInfo</code> from other XML structures.
317     *
318     * @param content a list of one or more {@link XMLStructure}s representing
319     *     key information types. The list is defensively copied to protect
320     *     against subsequent modification.
321     * @param id the value of an XML <code>ID</code> (may be <code>>null</code>)
322     * @return a <code>KeyInfo</code>
323     * @throws NullPointerException if <code>content</code> is <code>>null</code>
324     * @throws IllegalArgumentException if <code>content</code> is empty
325     * @throws ClassCastException if <code>content</code> contains any entries
326     *     that are not of type {@link XMLStructure}
327     */
328     @SuppressWarnings("rawtypes")
329     public abstract KeyInfo newKeyInfo(List content, String id);

```

```

330
331 /**
332  * Creates a <code>KeyName</code> from the specified name.
333  *
334  * @param name the name that identifies the key
335  * @return a <code>KeyName</code>
336  * @throws NullPointerException if <code>name</code> is <code>>null</code>
337  */
338 public abstract KeyName newKeyName(String name);
339
340 /**
341  * Creates a <code>KeyValue</code> from the specified public key.
342  *
343  * @param key the public key
344  * @return a <code>KeyValue</code>
345  * @throws KeyException if the <code>key</code>'s algorithm is not
346  *     recognized or supported by this <code>KeyInfoFactory</code>
347  * @throws NullPointerException if <code>key</code> is <code>>null</code>
348  */
349 public abstract KeyValue newKeyValue(PublicKey key) throws KeyException;
350
351 /**
352  * Creates a <code>PGPData</code> from the specified PGP public key
353  * identifier.
354  *
355  * @param keyId a PGP public key identifier as defined in <a href=
356  *     "http://www.ietf.org/rfc/rfc2440.txt">RFC 2440</a>, section 11.2.
357  *     The array is cloned to protect against subsequent modification.
358  * @return a <code>PGPData</code>
359  * @throws NullPointerException if <code>keyId</code> is <code>>null</code>
360  * @throws IllegalArgumentException if the key id is not in the correct
361  *     format
362  */
363 public abstract PGPData newPGPData(byte[] keyId);
364
365 /**
366  * Creates a <code>PGPData</code> from the specified PGP public key
367  * identifier, and optional key material packet and list of external
368  * elements.
369  *
370  * @param keyId a PGP public key identifier as defined in <a href=
371  *     "http://www.ietf.org/rfc/rfc2440.txt">RFC 2440</a>, section 11.2.
372  *     The array is cloned to protect against subsequent modification.
373  * @param keyPacket a PGP key material packet as defined in <a href=
374  *     "http://www.ietf.org/rfc/rfc2440.txt">RFC 2440</a>, section 5.5.
375  *     The array is cloned to protect against subsequent modification. May
376  *     be <code>>null</code>.
377  * @param other a list of {@link XMLStructure}s representing elements from
378  *     an external namespace. The list is defensively copied to protect
379  *     against subsequent modification. May be <code>>null</code> or empty.
380  * @return a <code>PGPData</code>
381  * @throws NullPointerException if <code>keyId</code> is <code>>null</code>
382  * @throws IllegalArgumentException if the <code>keyId</code> or

```



```

383 * <code>keyPacket</code> is not in the correct format. For
384 * <code>keyPacket</code>, the format of the packet header is
385 * checked and the tag is verified that it is of type key material. The
386 * contents and format of the packet body are not checked.
387 * @throws ClassCastException if <code>other</code> contains any
388 * entries that are not of type {@link XMLStructure}
389 */
390 @SuppressWarnings("rawtypes")
391 public abstract PGPData newPGPData(byte[] keyId, byte[] keyPacket,
392     List other);
393
394 /**
395 * Creates a <code>PGPData</code> from the specified PGP key material
396 * packet and optional list of external elements.
397 *
398 * @param keyPacket a PGP key material packet as defined in <a href=
399 *     "http://www.ietf.org/rfc/rfc2440.txt">RFC 2440</a>, section 5.5.
400 * The array is cloned to protect against subsequent modification.
401 * @param other a list of {@link XMLStructure}s representing elements from
402 * an external namespace. The list is defensively copied to protect
403 * against subsequent modification. May be <code>null</code> or empty.
404 * @return a <code>PGPData</code>
405 * @throws NullPointerException if <code>keyPacket</code> is
406 * <code>null</code>
407 * @throws IllegalArgumentException if <code>keyPacket</code> is not in the
408 * correct format. For <code>keyPacket</code>, the format of the packet
409 * header is checked and the tag is verified that it is of type key
410 * material. The contents and format of the packet body are not checked.
411 * @throws ClassCastException if <code>other</code> contains any
412 * entries that are not of type {@link XMLStructure}
413 */
414 @SuppressWarnings("rawtypes")
415 public abstract PGPData newPGPData(byte[] keyPacket, List other);
416
417 /**
418 * Creates a <code>RetrievalMethod</code> from the specified URI.
419 *
420 * @param uri the URI that identifies the <code>KeyInfo</code> information
421 * to be retrieved
422 * @return a <code>RetrievalMethod</code>
423 * @throws NullPointerException if <code>uri</code> is <code>null</code>
424 * @throws IllegalArgumentException if <code>uri</code> is not RFC 2396
425 * compliant
426 */
427 public abstract RetrievalMethod newRetrievalMethod(String uri);
428
429 /**
430 * Creates a <code>RetrievalMethod</code> from the specified parameters.
431 *
432 * @param uri the URI that identifies the <code>KeyInfo</code> information
433 * to be retrieved
434 * @param type a URI that identifies the type of <code>KeyInfo</code>
435 * information to be retrieved (may be <code>null</code>)

```

```

436 * @param transforms a list of {@link Transform}s. The list is defensively
437 *   copied to protect against subsequent modification. May be
438 *   <code>null</code> or empty.
439 * @return a <code>RetrievalMethod</code>
440 * @throws NullPointerException if <code>uri</code> is <code>null</code>
441 * @throws IllegalArgumentException if <code>uri</code> is not RFC 2396
442 *   compliant
443 * @throws ClassCastException if <code>transforms</code> contains any
444 *   entries that are not of type {@link Transform}
445 */
446 @SuppressWarnings("rawtypes")
447 public abstract RetrievalMethod newRetrievalMethod(String uri, String type,
448   List transforms);
449
450 /**
451 * Creates a <code>X509Data</code> containing the specified list of
452 * X.509 content.
453 *
454 * @param content a list of one or more X.509 content types. Valid types are
455 *   {@link String} (subject names), <code>byte[]</code> (subject key ids),
456 *   {@link java.security.cert.X509Certificate}, {@link X509CRL},
457 *   or {@link XMLStructure} ({@link X509IssuerSerial}
458 *   objects or elements from an external namespace). Subject names are
459 *   distinguished names in RFC 2253 String format. Implementations MUST
460 *   support the attribute type keywords defined in RFC 2253 (CN, L, ST,
461 *   O, OU, C, STREET, DC and UID). Implementations MAY support additional
462 *   keywords. The list is defensively copied to protect against
463 *   subsequent modification.
464 * @return a <code>X509Data</code>
465 * @throws NullPointerException if <code>content</code> is <code>null</code>
466 * @throws IllegalArgumentException if <code>content</code> is empty, or
467 *   if a subject name is not RFC 2253 compliant or one of the attribute
468 *   type keywords is not recognized.
469 * @throws ClassCastException if <code>content</code> contains any entries
470 *   that are not of one of the valid types mentioned above
471 */
472 @SuppressWarnings("rawtypes")
473 public abstract X509Data newX509Data(List content);
474
475 /**
476 * Creates an <code>X509IssuerSerial</code> from the specified X.500 issuer
477 * distinguished name and serial number.
478 *
479 * @param issuerName the issuer's distinguished name in RFC 2253 String
480 *   format. Implementations MUST support the attribute type keywords
481 *   defined in RFC 2253 (CN, L, ST, O, OU, C, STREET, DC and UID).
482 *   Implementations MAY support additional keywords.
483 * @param serialNumber the serial number
484 * @return an <code>X509IssuerSerial</code>
485 * @throws NullPointerException if <code>issuerName</code> or
486 *   <code>serialNumber</code> are <code>null</code>
487 * @throws IllegalArgumentException if the issuer name is not RFC 2253
488 *   compliant or one of the attribute type keywords is not recognized.

```

```

489     */
490     public abstract X509IssuerSerial newX509IssuerSerial
491         (String issuerName, BigInteger serialNumber);
492
493     /**
494     * Indicates whether a specified feature is supported.
495     *
496     * @param feature the feature name (as an absolute URI)
497     * @return <code>true</code> if the specified feature is supported,
498     *         <code>false</code> otherwise
499     * @throws NullPointerException if <code>feature</code> is <code>null</code>
500     */
501     public abstract boolean isFeatureSupported(String feature);
502
503     /**
504     * Returns a reference to the <code>URIDereferencer</code> that is used by
505     * default to dereference URIs in {@link RetrievalMethod} objects.
506     *
507     * @return a reference to the default <code>URIDereferencer</code>
508     */
509     public abstract URIDereferencer getURIDereferencer();
510
511     /**
512     * Unmarshals a new <code>KeyInfo</code> instance from a
513     * mechanism-specific <code>XMLStructure</code> (ex: {@link DOMStructure})
514     * instance.
515     *
516     * @param xmlStructure a mechanism-specific XML structure from which to
517     *         unmarshal the keyinfo from
518     * @return the <code>KeyInfo</code>
519     * @throws NullPointerException if <code>xmlStructure</code> is
520     *         <code>null</code>
521     * @throws ClassCastException if the type of <code>xmlStructure</code> is
522     *         inappropriate for this factory
523     * @throws MarshalException if an unrecoverable exception occurs during
524     *         unmarshalling
525     */
526     public abstract KeyInfo unmarshalKeyInfo(XMLStructure xmlStructure)
527         throws MarshalException;
528 }

```

## 12.3 KeyName.java

Secuencia 129: javax/xml/crypto/dsig/keyinfo/KeyName.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```

```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: KeyName.java,v 1.4 2005/05/10 16:35:35 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.keyinfo;
29
30  import javax.xml.crypto.XMLStructure;
31
32  /**
33   * A representation of the XML KeyName element as
34   * defined in the http://www.w3.org/TR/xmlsig-core/.
35   * W3C Recommendation for XML-Signature Syntax and Processing.
36   * A KeyName object contains a string value which may be used
37   * by the signer to communicate a key identifier to the recipient. The
38   * XML Schema Definition is defined as:
39   *
40   * <pre>
41   * <?element name="KeyName" type="string"/>
42   * </pre>
43   *
44   * A KeyName instance may be created by invoking the
45   * {@link KeyInfoFactory#newKeyName newKeyName} method of the
46   * {@link KeyInfoFactory} class, and passing it a String
47   * representing the name of the key; for example:
48   * <pre>
49   * KeyInfoFactory factory = KeyInfoFactory.getInstance("DOM");
50   * KeyName keyName = factory.newKeyName("Alice");
51   * </pre>
52   *
53   * @author Sean Mullan
54   * @author JSR 105 Expert Group
55   * @since 1.6
56   * @see KeyInfoFactory#newKeyName(String)
57   */
58  public interface KeyName extends XMLStructure {
59
60      /**
61       * Returns the name of this KeyName.
62       *

```

```

63     * @return the name of this <code>KeyName</code> (never
64     *     <code>null</code>)
65     */
66     String getName();
67 }

```

## 12.4 KeyValue.java

Secuencia 130: javax/xml/crypto/dsig/keyinfo/KeyValue.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: KeyValue.java,v 1.4 2005/05/10 16:35:35 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.keyinfo;
29
30 import java.security.KeyException;
31 import java.security.KeyStore;
32 import java.security.PublicKey;
33 import java.security.interfaces.DSAPublicKey;
34 import java.security.interfaces.RSAPublicKey;
35 import javax.xml.crypto.XMLStructure;
36
37 /**
38 * A representation of the XML <code>KeyValue</code> element as defined
39 * in the <a href="http://www.w3.org/TR/xmlldsig-core/">
40 * W3C Recommendation for XML-Signature Syntax and Processing</a>. A
41 * <code>KeyValue</code> object contains a single public key that may be
42 * useful in validating the signature. The XML schema definition is defined as:
43 *
44 * <pre>

```

```

45 *      <element name="KeyValue" type="ds:KeyValue"/>
46 *      <complexType name="KeyValue" mixed="true"/>
47 *      <choice>
48 *          <element ref="ds:DSAKeyValue"/>
49 *          <element ref="ds:RSAKeyValue"/>
50 *          <any namespace="##other" processContents="lax"/>
51 *      </choice>
52 *      </complexType>
53 *
54 *      <element name="DSAKeyValue" type="ds:DSAKeyValue"/>
55 *      <complexType name="DSAKeyValue">
56 *          <sequence>
57 *              <sequence minOccurs="0">
58 *                  <element name="P" type="ds:CryptoBinary"/>
59 *                  <element name="Q" type="ds:CryptoBinary"/>
60 *              </sequence>
61 *              <element name="G" type="ds:CryptoBinary" minOccurs="0"/>
62 *              <element name="Y" type="ds:CryptoBinary"/>
63 *              <element name="J" type="ds:CryptoBinary" minOccurs="0"/>
64 *              <sequence minOccurs="0">
65 *                  <element name="Seed" type="ds:CryptoBinary"/>
66 *                  <element name="PgenCounter" type="ds:CryptoBinary"/>
67 *              </sequence>
68 *          </sequence>
69 *      </complexType>
70 *
71 *      <element name="RSAKeyValue" type="ds:RSAKeyValue"/>
72 *      <complexType name="RSAKeyValue">
73 *          <sequence>
74 *              <element name="Modulus" type="ds:CryptoBinary"/>
75 *              <element name="Exponent" type="ds:CryptoBinary"/>
76 *          </sequence>
77 *      </complexType>
78 * </pre>
79 * A <code>KeyValue</code> instance may be created by invoking the
80 * {@link KeyInfoFactory#newKeyValue newKeyValue} method of the
81 * {@link KeyInfoFactory} class, and passing it a {@link
82 * java.security.PublicKey} representing the value of the public key. Here is
83 * an example of creating a <code>KeyValue</code> from a {@link DSAPublicKey}
84 * of a {@link java.security.cert.Certificate} stored in a
85 * {@link java.security.KeyStore}:
86 * <pre>
87 * KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
88 * PublicKey dsaPublicKey = keyStore.getCertificate("myDSASigningCert").getPublicKey();
89 * KeyInfoFactory factory = KeyInfoFactory.getInstance("DOM");
90 * KeyValue keyValue = factory.newKeyValue(dsaPublicKey);
91 * </pre>
92 *
93 * This class returns the <code>DSAKeyValue</code> and
94 * <code>RSAKeyValue</code> elements as objects of type
95 * {@link DSAPublicKey} and {@link RSAPublicKey}, respectively. Note that not
96 * all of the fields in the schema are accessible as parameters of these
97 * types.

```

```

98  *
99  * @author Sean Mullan
100 * @author JSR 105 Expert Group
101 * @since 1.6
102 * @see KeyInfoFactory#newKeyValue(PublicKey)
103 */
104 public interface KeyValue extends XMLStructure {
105
106     /**
107      * URI identifying the DSA KeyValue KeyInfo type:
108      * http://www.w3.org/2000/09/xmldsig#DSAKeyValue. This can be specified as
109      * the value of the <code>type</code> parameter of the
110      * {@link RetrievalMethod} class to describe a remote
111      * <code>DSAKeyValue</code> structure.
112      */
113     final static String DSA_TYPE =
114         "http://www.w3.org/2000/09/xmldsig#DSAKeyValue";
115
116     /**
117      * URI identifying the RSA KeyValue KeyInfo type:
118      * http://www.w3.org/2000/09/xmldsig#RSAKeyValue. This can be specified as
119      * the value of the <code>type</code> parameter of the
120      * {@link RetrievalMethod} class to describe a remote
121      * <code>RSAKeyValue</code> structure.
122      */
123     final static String RSA_TYPE =
124         "http://www.w3.org/2000/09/xmldsig#RSAKeyValue";
125
126     /**
127      * Returns the public key of this <code>KeyValue</code>.
128      *
129      * @return the public key of this <code>KeyValue</code>
130      * @throws KeyException if this <code>KeyValue</code> cannot be converted
131      *         to a <code>PublicKey</code>
132      */
133     PublicKey getPublicKey() throws KeyException;
134 }

```

## 12.5 PGPDData.java

Secuencia 131: javax/xml/crypto/dsig/keyinfo/PGPDData.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```

13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: PGPDData.java,v 1.4 2005/05/10 16:35:35 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.keyinfo;
29
30  import java.util.Collections;
31  import java.util.List;
32  import javax.xml.crypto.XMLStructure;
33
34  /**
35   * A representation of the XML <code>PGPDData</code> element as defined in
36   * the <a href="http://www.w3.org/TR/xmlsig-core/">
37   * W3C Recommendation for XML-Signature Syntax and Processing</a>. A
38   * <code>PGPDData</code> object is used to convey information related to
39   * PGP public key pairs and signatures on such keys. The XML Schema Definition
40   * is defined as:
41   *
42   * <pre>
43   *   <lt;element name="PGPDData" type="ds:PGPDDataType"/>
44   *   <lt;complexType name="PGPDDataType">
45   *     <lt;choice>
46   *       <lt;sequence>
47   *         <lt;element name="PGPKeyID" type="base64Binary"/>
48   *         <lt;element name="PGPKeyPacket" type="base64Binary" minOccurs="0"/>
49   *         <lt;any namespace="##other" processContents="lax" minOccurs="0"
50   *           maxOccurs="unbounded"/>
51   *       </sequence>
52   *     </choice>
53   *   </complexType>
54   * </pre>
55   *
56   * A <code>PGPDData</code> instance may be created by invoking one of the
57   * {@link KeyInfoFactory#newPGPDData newPGPDData} methods of the {@link
58   * KeyInfoFactory} class, and passing it
59   * <code>byte</code> arrays representing the contents of the PGP public key
60   * identifier and/or PGP key material packet, and an optional list of

```



```

66  * elements from an external namespace.
67  *
68  * @author Sean Mullan
69  * @author JSR 105 Expert Group
70  * @since 1.6
71  * @see KeyInfoFactory#newPGPDData(byte[])
72  * @see KeyInfoFactory#newPGPDData(byte[], byte[], List)
73  * @see KeyInfoFactory#newPGPDData(byte[], List)
74  */
75  public interface PGPDData extends XMLStructure {
76
77      /**
78       * URI identifying the PGPDData KeyInfo type:
79       * http://www.w3.org/2000/09/xmldsig#PGPDData. This can be specified as the
80       * value of the <code>type</code> parameter of the {@link RetrievalMethod}
81       * class to describe a remote <code>PGPDData</code> structure.
82       */
83      final static String TYPE = "http://www.w3.org/2000/09/xmldsig#PGPDData";
84
85      /**
86       * Returns the PGP public key identifier of this <code>PGPDData</code> as
87       * defined in <a href="http://www.ietf.org/rfc/rfc2440.txt">RFC 2440</a>,
88       * section 11.2.
89       *
90       * @return the PGP public key identifier (may be <code>null</code> if
91       *         not specified). Each invocation of this method returns a new clone
92       *         to protect against subsequent modification.
93       */
94      byte[] getKeyId();
95
96      /**
97       * Returns the PGP key material packet of this <code>PGPDData</code> as
98       * defined in <a href="http://www.ietf.org/rfc/rfc2440.txt">RFC 2440</a>,
99       * section 5.5.
100      *
101      * @return the PGP key material packet (may be <code>null</code> if not
102      *         specified). Each invocation of this method returns a new clone to
103      *         protect against subsequent modification.
104      */
105      byte[] getKeyPacket();
106
107      /**
108       * Returns an {@link Collections#unmodifiableList unmodifiable list}
109       * of {@link XMLStructure}s representing elements from an external
110       * namespace.
111       *
112       * @return an unmodifiable list of <code>XMLStructure</code>s (may be
113       *         empty, but never <code>null</code>)
114       */
115      @SuppressWarnings("rawtypes")
116      List getExternalElements();
117  }

```

## 12.6 RetrievalMethod.java

Secuencia 132: javax/xml/crypto/dsig/keyinfo/RetrievalMethod.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: RetrievalMethod.java,v 1.8 2005/05/10 16:35:35 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.keyinfo;
29
30 import javax.xml.crypto.Data;
31 import javax.xml.crypto.URIReference;
32 import javax.xml.crypto.URIReferenceException;
33 import javax.xml.crypto.XMLCryptoContext;
34 import javax.xml.crypto.XMLStructure;
35 import javax.xml.crypto.dsig.Transform;
36 import java.util.List;
37
38 /**
39 * A representation of the XML RetrievalMethod element as
40 * defined in the http://www.w3.org/TR/xmlldsig-core/
41 * W3C Recommendation for XML-Signature Syntax and Processing.
42 * A RetrievalMethod object is used to convey a reference to
43 * KeyInfo information that is stored at another location.
44 * The XML schema definition is defined as:
45 *
46 * <pre>
47 *   <element name="RetrievalMethod" type="ds:RetrievalMethodType"/>
48 *   <complexType name="RetrievalMethodType">
49 *     <sequence>
50 *       <element name="Transforms" type="ds:TransformsType" minOccurs="0"/>
```

```

51 *      <sequence>
52 *      <attribute name="URI" type="anyURI"/>
53 *      <attribute name="Type" type="anyURI" use="optional"/>
54 *      </complexType>
55 * </pre>
56 *
57 * A <code>RetrievalMethod</code> instance may be created by invoking one of the
58 * {@link KeyInfoFactory#newRetrievalMethod newRetrievalMethod} methods
59 * of the {@link KeyInfoFactory} class, and passing it the URI
60 * identifying the location of the KeyInfo, an optional type URI identifying
61 * the type of KeyInfo, and an optional list of {@link Transform}s; for example:
62 * <pre>
63 *   KeyInfoFactory factory = KeyInfoFactory.getInstance("DOM");
64 *   RetrievalMethod rm = factory.newRetrievalMethod
65 *       ("#KeyValue-1", KeyValue.DSA_TYPE, Collections.singletonList(Transform.BASE64));
66 * </pre>
67 *
68 * @author Sean Mullan
69 * @author JSR 105 Expert Group
70 * @since 1.6
71 * @see KeyInfoFactory#newRetrievalMethod(String)
72 * @see KeyInfoFactory#newRetrievalMethod(String, String, List)
73 */
74 public interface RetrievalMethod extends URIReference, XMLStructure {
75
76     /**
77      * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
78      * list} of {@link Transform}s of this <code>RetrievalMethod</code>.
79      *
80      * @return an unmodifiable list of <code>Transform</code> objects (may be
81      *         empty but never <code>null</code>).
82      */
83     @SuppressWarnings("rawtypes")
84     List getTransforms();
85
86     /**
87      * Returns the URI of the referenced <code>KeyInfo</code> information.
88      *
89      * @return the URI of the referenced <code>KeyInfo</code> information in
90      *         RFC 2396 format (never <code>null</code>)
91      */
92     String getURI();
93
94     /**
95      * Dereferences the <code>KeyInfo</code> information referenced by this
96      * <code>RetrievalMethod</code> and applies the specified
97      * <code>Transform</code>s.
98      *
99      * @param context an <code>XMLCryptoContext</code> that may contain
100      *        additional useful information for dereferencing the URI. The
101      *        context's <code>baseURI</code> and <code>dereferencer</code>
102      *        parameters (if specified) are used to resolve and dereference this
103      *        <code>RetrievalMethod</code>

```

```

104 * @return a Data object representing the raw contents of the
105 *   KeyInfo information referenced by this
106 *   RetrievalMethod. It is the caller's responsibility to
107 *   convert the returned data to an appropriate
108 *   KeyInfo object.
109 * @throws NullPointerException if context is null
110 * @throws URIReferenceException if there is an error while dereferencing
111 */
112 Data dereference(XMLCryptoContext context) throws URIReferenceException;
113 }

```

## 12.7 X509Data.java

Secuencia 133: javax/xml/crypto/dsig/keyinfo/X509Data.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: X509Data.java,v 1.4 2005/05/10 16:35:35 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.keyinfo;
29
30 import javax.xml.crypto.XMLStructure;
31 import java.security.cert.X509CRL;
32 import java.util.List;
33
34 /**
35 * A representation of the XML X509Data element as defined in
36 * the http://www.w3.org/TR/xmlldsig-core/. An
37 * W3C Recommendation for XML-Signature Syntax and Processing. An
38 * X509Data object contains one or more identifiers of keys
39 * or X.509 certificates (or certificates' identifiers or a revocation list).

```

```

40 * The XML Schema Definition is defined as:
41 *
42 * <pre>
43 *     <element name="X509Data" type="ds:X509DataType"/>;
44 *     <complexType name="X509DataType">;
45 *         <sequence maxOccurs="unbounded">;
46 *             <choice>;
47 *                 <element name="X509IssuerSerial" type="ds:X509IssuerSerialType"/>;
48 *                 <element name="X509SKI" type="base64Binary"/>;
49 *                 <element name="X509SubjectName" type="string"/>;
50 *                 <element name="X509Certificate" type="base64Binary"/>;
51 *                 <element name="X509CRL" type="base64Binary"/>;
52 *                 <any namespace="##other" processContents="lax"/>;
53 *             </choice>;
54 *         </sequence>;
55 *     </complexType>;
56 *
57 *     <complexType name="X509IssuerSerialType">;
58 *         <sequence>;
59 *             <element name="X509IssuerName" type="string"/>;
60 *             <element name="X509SerialNumber" type="integer"/>;
61 *         </sequence>;
62 *     </complexType>;
63 * </pre>
64 *
65 * An <code>X509Data</code> instance may be created by invoking the
66 * {@link KeyInfoFactory#newX509Data newX509Data} methods of the
67 * {@link KeyInfoFactory} class and passing it a list of one or more
68 * {@link XMLStructure}s representing X.509 content; for example:
69 * <pre>
70 *     KeyInfoFactory factory = KeyInfoFactory.getInstance("DOM");
71 *     X509Data x509Data = factory.newX509Data
72 *         (Collections.singletonList("cn=Alice"));
73 * </pre>
74 *
75 * @author Sean Mullan
76 * @author JSR 105 Expert Group
77 * @since 1.6
78 * @see KeyInfoFactory#newX509Data(List)
79 */
80 //@@@ check for illegal combinations of data violating MUSTs in W3c spec
81 public interface X509Data extends XMLStructure {
82
83     /**
84      * URI identifying the X509Data KeyInfo type:
85      * http://www.w3.org/2000/09/xmldsig#X509Data. This can be specified as
86      * the value of the <code>type</code> parameter of the
87      * {@link RetrievalMethod} class to describe a remote
88      * <code>X509Data</code> structure.
89      */
90     final static String TYPE = "http://www.w3.org/2000/09/xmldsig#X509Data";
91
92     /**

```

```

93      * URI identifying the binary (ASN.1 DER) X.509 Certificate KeyInfo type:
94      * http://www.w3.org/2000/09/xmlsig#rawX509Certificate. This can be
95      * specified as the value of the <code>type</code> parameter of the
96      * {@link RetrievalMethod} class to describe a remote X509 Certificate.
97      */
98      final static String RAW_X509_CERTIFICATE_TYPE =
99          "http://www.w3.org/2000/09/xmlsig#rawX509Certificate";
100
101      /**
102       * Returns an {@link java.util.Collections#unmodifiableList unmodifiable
103       * list} of the content in this <code>X509Data</code>. Valid types are
104       * {@link String} (subject names), <code>byte[]</code> (subject key ids),
105       * {@link java.security.cert.X509Certificate}, {@link X509CRL},
106       * or {@link XMLStructure} ({@link X509IssuerSerial}
107       * objects or elements from an external namespace).
108       *
109       * @return an unmodifiable list of the content in this <code>X509Data</code>
110       *         (never <code>null</code> or empty)
111       */
112      @SuppressWarnings("rawtypes")
113      List getContent();
114  }

```

## 12.8 X509IssuerSerial.java

Secuencia 134: javax/xml/crypto/dsig/keyinfo/X509IssuerSerial.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: X509IssuerSerial.java,v 1.4 2005/05/10 16:35:35 mullan Exp $
27 */

```

```

28 package javax.xml.crypto.dsig.keyinfo;
29
30 import java.math.BigInteger;
31 import java.security.cert.X509Certificate;
32 import javax.xml.crypto.XMLStructure;
33
34 /**
35  * A representation of the XML X509IssuerSerial element as
36  * defined in the http://www.w3.org/TR/xmlldsig-core/
37  * W3C Recommendation for XML-Signature Syntax and Processing.
38  * An X509IssuerSerial object contains an X.509 issuer
39  * distinguished name (DN) and serial number pair. The XML schema definition is
40  * defined as:
41  *
42  * <pre>
43  *   <element name="X509IssuerSerial" type="ds:X509IssuerSerialType"/>
44  *   <complexType name="X509IssuerSerialType">
45  *     <sequence>
46  *       <element name="X509IssuerName" type="string"/>
47  *       <element name="X509SerialNumber" type="integer"/>
48  *     </sequence>
49  *   </complexType>
50  * </pre>
51  *
52  * An X509IssuerSerial instance may be created by invoking the
53  * {@link KeyInfoFactory#newX509IssuerSerial newX509IssuerSerial} method
54  * of the {@link KeyInfoFactory} class, and passing it a
55  * String and BigInteger representing the X.509
56  * DN and serial number. Here is an example of creating an
57  * X509IssuerSerial from the issuer DN and serial number of an
58  * existing {@link X509Certificate}:
59  * <pre>
60  * KeyInfoFactory factory = KeyInfoFactory.getInstance("DOM");
61  * X509IssuerSerial issuer = factory.newX509IssuerSerial
62  *   (cert.getIssuerX500Principal().getName(), cert.getSerialNumber());
63  * </pre>
64  *
65  * @author Sean Mullan
66  * @author JSR 105 Expert Group
67  * @since 1.6
68  * @see X509Data#getContent
69  * @see KeyInfoFactory#newX509IssuerSerial(String, BigInteger)
70  */
71 public interface X509IssuerSerial extends XMLStructure {
72
73     /**
74      * Returns the X.509 distinguished name of this
75      * X509IssuerSerial in
76      * http://www.ietf.org/rfc/rfc2253.txt RFC 2253 String format.
77      *
78      * @return the X.509 distinguished name in RFC 2253 String format (never
79      *   null)
80      */

```

```

81     String getIssuerName();
82
83     /**
84      * Returns the serial number of this <code>X509IssuerSerial</code>.
85      *
86      * @return the serial number (never <code>null</code>)
87      */
88     BigInteger getSerialNumber();
89 }

```

## 13 Xml Crypto Dsig Spec (javax.xml.crypto.dsig.spec package)

### 13.1 C14NMethodParameterSpec.java

Secuencia 135: javax.xml.crypto.dsig.spec/C14NMethodParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: C14NMethodParameterSpec.java,v 1.3 2005/05/10 16:40:17 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.spec;
29
30 import javax.xml.crypto.dsig.CanonicalizationMethod;
31
32 /**
33 * A specification of algorithm parameters for a {@link CanonicalizationMethod}
34 * Algorithm. The purpose of this interface is to group (and provide type
35 * safety for) all canonicalization (C14N) parameter specifications. All
36 * canonicalization parameter specifications must implement this interface.
37 *
38 * @author Sean Mullan

```



```

39  * @author JSR 105 Expert Group
40  * @since 1.6
41  * @see CanonicalizationMethod
42  */
43  public interface C14NMethodParameterSpec extends TransformParameterSpec {}

```

## 13.2 DigestMethodParameterSpec.java

Secuencia 136: javax/xml/crypto/dsig/spec/DigestMethodParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: DigestMethodParameterSpec.java,v 1.3 2005/05/10 16:40:17 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.spec;
29
30 import javax.xml.crypto.dsig.DigestMethod;
31 import java.security.spec.AlgorithmParameterSpec;
32
33 /**
34  * A specification of algorithm parameters for a {@link DigestMethod}
35  * algorithm. The purpose of this interface is to group (and provide type
36  * safety for) all digest method parameter specifications. All digest method
37  * parameter specifications must implement this interface.
38  *
39  * @author Sean Mullan
40  * @author JSR 105 Expert Group
41  * @since 1.6
42  * @see DigestMethod
43  */
44 public interface DigestMethodParameterSpec extends AlgorithmParameterSpec {}

```

### 13.3 ExcC14NParameterSpec.java

Secuencia 137: javax/xml/crypto/dsig/spec/ExcC14NParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: ExcC14NParameterSpec.java,v 1.7 2005/05/13 18:45:42 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.spec;
29
30 import javax.xml.crypto.dsig.CanonicalizationMethod;
31 import java.util.ArrayList;
32 import java.util.Collections;
33 import java.util.List;
34
35 /**
36 * Parameters for the W3C Recommendation:
37 * <a href="http://www.w3.org/TR/xml-exc-c14n/">
38 * Exclusive XML Canonicalization (C14N) algorithm</a>. The
39 * parameters include an optional inclusive namespace prefix list. The XML
40 * Schema Definition of the Exclusive XML Canonicalization parameters is
41 * defined as:
42 * <pre><code>
43 * <?xml schema xmlns="http://www.w3.org/2001/XMLSchema"
44 *   xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"
45 *   targetNamespace="http://www.w3.org/2001/10/xml-exc-c14n#"
46 *   version="0.1" elementFormDefault="qualified">
47 *
48 *   <element name="InclusiveNamespaces" type="ec:InclusiveNamespaces"/>
49 *   <complexType name="InclusiveNamespaces">
50 *     <attribute name="PrefixList" type="xsd:string"/>

```

```

51 * <code></code></pre>
52 * <code></code></pre>
53 * </code></pre>
54 *
55 * @author Sean Mullan
56 * @author JSR 105 Expert Group
57 * @since 1.6
58 * @see CanonicalizationMethod
59 */
60 public final class ExcC14NParameterSpec implements C14NMethodParameterSpec {
61
62     private List<String> preList;
63
64     /**
65      * Indicates the default namespace ("#default").
66      */
67     public static final String DEFAULT = "#default";
68
69     /**
70      * Creates a <code>ExcC14NParameterSpec</code> with an empty prefix
71      * list.
72      */
73     public ExcC14NParameterSpec() {
74         preList = Collections.emptyList();
75     }
76
77     /**
78      * Creates a <code>ExcC14NParameterSpec</code> with the specified list
79      * of prefixes. The list is copied to protect against subsequent
80      * modification.
81      *
82      * @param prefixList the inclusive namespace prefix list. Each entry in
83      *     the list is a <code>String</code> that represents a namespace prefix.
84      * @throws NullPointerException if <code>prefixList</code> is
85      *     <code>null</code>
86      * @throws ClassCastException if any of the entries in the list are not
87      *     of type <code>String</code>
88      */
89     @SuppressWarnings("rawtypes")
90     public ExcC14NParameterSpec(List prefixList) {
91         if (prefixList == null) {
92             throw new NullPointerException("prefixList cannot be null");
93         }
94         List<?> copy = new ArrayList<>((List<?>)prefixList);
95         for (int i = 0, size = copy.size(); i < size; i++) {
96             if (!(copy.get(i) instanceof String)) {
97                 throw new ClassCastException("not a String");
98             }
99         }
100
101         @SuppressWarnings("unchecked")
102         List<String> temp = (List<String>)copy;
103

```

```

104     preList = Collections.unmodifiableList(temp);
105 }
106
107 /**
108  * Returns the inclusive namespace prefix list. Each entry in the list
109  * is a String that represents a namespace prefix.
110  *
111  * 

This implementation returns an {@link
112  * java.util.Collections#unmodifiableList unmodifiable list}.
113  *
114  * @return the inclusive namespace prefix list (may be empty but never
115  *         null)
116  */
117 @SuppressWarnings("rawtypes")
118 public List getPrefixList() {
119     return preList;
120 }
121 }


```

### 13.4 HMACParameterSpec.java

Secuencia 138: javax/xml/crypto/dsig/spec/HMACParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: HMACParameterSpec.java,v 1.4 2005/05/10 16:40:17 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.spec;
29
30 import javax.xml.crypto.dsig.SignatureMethod;
31

```

```

32 /**
33  * Parameters for the <a href="http://www.w3.org/TR/xmlsig-core/#sec-MACs">
34  * XML Signature HMAC Algorithm</a>. The parameters include an optional output
35  * length which specifies the MAC truncation length in bits. The resulting
36  * HMAC will be truncated to the specified number of bits. If the parameter is
37  * not specified, then this implies that all the bits of the hash are to be
38  * output. The XML Schema Definition of the <code>HMACOutputLength</code>
39  * element is defined as:
40  * <pre><code>
41  * <lt;element name="HMACOutputLength" minOccurs="0" type="ds:HMACOutputLengthType"/>
42  * <lt;simpleType name="HMACOutputLengthType">
43  *   <lt;restriction base="integer"/>
44  * <lt;/simpleType>
45  * </code></pre>
46  *
47  * @author Sean Mullan
48  * @author JSR 105 Expert Group
49  * @since 1.6
50  * @see SignatureMethod
51  * @see <a href="http://www.ietf.org/rfc/rfc2104.txt">RFC 2104</a>
52  */
53 public final class HMACParameterSpec implements SignatureMethodParameterSpec {
54
55     private int outputLength;
56
57     /**
58      * Creates an <code>HMACParameterSpec</code> with the specified truncation
59      * length.
60      *
61      * @param outputLength the truncation length in number of bits
62      */
63     public HMACParameterSpec(int outputLength) {
64         this.outputLength = outputLength;
65     }
66
67     /**
68      * Returns the truncation length.
69      *
70      * @return the truncation length in number of bits
71      */
72     public int getOutputLength() {
73         return outputLength;
74     }
75 }

```

### 13.5 SignatureMethodParameterSpec.java

Secuencia 139: javax/xml/crypto/dsig/spec/SignatureMethodParameterSpec.java

```

1 /**
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *

```

```

6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26  * $Id: SignatureMethodParameterSpec.java,v 1.3 2005/05/10 16:40:17 mullan Exp $
27  */
28 package javax.xml.crypto.dsig.spec;
29
30 import javax.xml.crypto.dsig.SignatureMethod;
31 import java.security.spec.AlgorithmParameterSpec;
32
33 /**
34  * A specification of algorithm parameters for an XML {@link SignatureMethod}
35  * algorithm. The purpose of this interface is to group (and provide type
36  * safety for) all signature method parameter specifications. All signature
37  * method parameter specifications must implement this interface.
38  *
39  * @author Sean Mullan
40  * @author JSR 105 Expert Group
41  * @since 1.6
42  * @see SignatureMethod
43  */
44 public interface SignatureMethodParameterSpec extends AlgorithmParameterSpec {}

```

## 13.6 TransformParameterSpec.java

Secuencia 140: javax/xml/crypto/dsig/spec/TransformParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```

```

11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: TransformParameterSpec.java,v 1.3 2005/05/10 16:40:17 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.spec;
29
30  import javax.xml.crypto.dsig.Transform;
31  import java.security.spec.AlgorithmParameterSpec;
32
33  /**
34   * A specification of algorithm parameters for a {@link Transform}
35   * algorithm. The purpose of this interface is to group (and provide type
36   * safety for) all transform parameter specifications. All transform parameter
37   * specifications must implement this interface.
38   *
39   * @author Sean Mullan
40   * @author JSR 105 Expert Group
41   * @since 1.6
42   * @see Transform
43   */
44  public interface TransformParameterSpec extends AlgorithmParameterSpec {}

```

### 13.7 XPathFilter2ParameterSpec.java

Secuencia 141: javax/xml/crypto/dsig/spec/XPathFilter2ParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: XPathFilter2ParameterSpec.java,v 1.7 2005/05/13 18:45:42 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.spec;
29
30  import java.util.ArrayList;
31  import java.util.Collections;
32  import java.util.List;
33  import javax.xml.crypto.dsig.Transform;
34
35  /**
36   * Parameters for the W3C Recommendation
37   * <a href="http://www.w3.org/TR/xmlldsig-filter2/">
38   * XPath Filter 2.0 Transform Algorithm</a>.
39   * The parameters include a list of one or more {@link XPathType} objects.
40   *
41   * @author Sean Mullan
42   * @author JSR 105 Expert Group
43   * @since 1.6
44   * @see Transform
45   * @see XPathFilterParameterSpec
46   */
47  public final class XPathFilter2ParameterSpec implements TransformParameterSpec {
48
49      private final List<XPathType> xPathList;
50
51      /**
52       * Creates an <code>XPathFilter2ParameterSpec</code>.
53       *
54       * @param xPathList a list of one or more {@link XPathType} objects. The
55       *     list is defensively copied to protect against subsequent modification.
56       * @throws ClassCastException if <code>xPathList</code> contains any
57       *     entries that are not of type {@link XPathType}
58       * @throws IllegalArgumentException if <code>xPathList</code> is empty
59       * @throws NullPointerException if <code>xPathList</code> is
60       *     <code>null</code>
61       */
62      @SuppressWarnings("rawtypes")
63      public XPathFilter2ParameterSpec(List xPathList) {
64          if (xPathList == null) {
65              throw new NullPointerException("xPathList cannot be null");
66          }
67          List<?> xPathListCopy = new ArrayList<>((List<?>)xPathList);
68          if (xPathListCopy.isEmpty()) {

```



```

69         throw new IllegalArgumentException("XPathList cannot be empty");
70     }
71     int size = XPathListCopy.size();
72     for (int i = 0; i < size; i++) {
73         if (!(XPathListCopy.get(i) instanceof XPathType)) {
74             throw new ClassCastException
75                 ("XPathList["+i+"] is not a valid type");
76         }
77     }
78
79     @SuppressWarnings("unchecked")
80     List<XPathType> temp = (List<XPathType>)XPathListCopy;
81
82     this.XPathList = Collections.unmodifiableList(temp);
83 }
84
85 /**
86  * Returns a list of one or more {@link XPathType} objects.
87  * <p>
88  * This implementation returns an {@link Collections#unmodifiableList
89  * unmodifiable list}.
90  *
91  * @return a <code>List</code> of <code>XPathType</code> objects
92  *         (never <code>null</code> or empty)
93  */
94 @SuppressWarnings("rawtypes")
95 public List getXPathList() {
96     return XPathList;
97 }
98 }

```

### 13.8 XPathFilterParameterSpec.java

Secuencia 142: javax/xml/crypto/dsig/spec/XPathFilterParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```

```

20  *
21  *
22  *
23  *
24  */
25  /*
26  * $Id: XPathFilterParameterSpec.java,v 1.4 2005/05/10 16:40:17 mullan Exp $
27  */
28  package javax.xml.crypto.dsig.spec;
29
30  import javax.xml.crypto.dsig.Transform;
31  import java.util.Collections;
32  import java.util.HashMap;
33  import java.util.Iterator;
34  import java.util.Map;
35  import java.util.Map.Entry;
36
37  /**
38   * Parameters for the <a href="http://www.w3.org/TR/xmldsig-core/#sec-XPath">
39   * XPath Filtering Transform Algorithm</a>.
40   * The parameters include the XPath expression and an optional <code>Map</code>
41   * of additional namespace prefix mappings. The XML Schema Definition of
42   * the XPath Filtering transform parameters is defined as:
43   * <pre><code>
44   * <lt;element name="XPath" type="string"/></code>
45   * </code></pre>
46   *
47   * @author Sean Mullan
48   * @author JSR 105 Expert Group
49   * @since 1.6
50   * @see Transform
51   */
52  public final class XPathFilterParameterSpec implements TransformParameterSpec {
53
54      private String xpath;
55      private Map<String,String> nsMap;
56
57      /**
58       * Creates an <code>XPathFilterParameterSpec</code> with the specified
59       * XPath expression.
60       *
61       * @param xpath the XPath expression to be evaluated
62       * @throws NullPointerException if <code>xpath</code> is <code>null</code>
63       */
64      public XPathFilterParameterSpec(String xpath) {
65          if (xpath == null) {
66              throw new NullPointerException();
67          }
68          this.xpath = xpath;
69          this.nsMap = Collections.emptyMap();
70      }
71
72      /**

```

```

73      * Creates an XPathFilterParameterSpec with the specified
74      * XPath expression and namespace map. The map is copied to protect against
75      * subsequent modification.
76      *
77      * @param xpath the XPath expression to be evaluated
78      * @param namespaceMap the map of namespace prefixes. Each key is a
79      *   namespace prefix String that maps to a corresponding
80      *   namespace URI String.
81      * @throws NullPointerException if xpath or
82      *   namespaceMap are null
83      * @throws ClassCastException if any of the map's keys or entries are not
84      *   of type String
85      */
86      @SuppressWarnings("rawtypes")
87      public XPathFilterParameterSpec(String xpath, Map namespaceMap) {
88          if (xpath == null || namespaceMap == null) {
89              throw new NullPointerException();
90          }
91          this.xpath = xpath;
92          Map<?, ?> copy = new HashMap<>((Map<?, ?>)namespaceMap);
93          Iterator<? extends Map.Entry<?, ?>> entries = copy.entrySet().iterator();
94          while (entries.hasNext()) {
95              Map.Entry<?, ?> me = entries.next();
96              if (!(me.getKey() instanceof String) ||
97                  !(me.getValue() instanceof String)) {
98                  throw new ClassCastException("not a String");
99              }
100          }
101
102          @SuppressWarnings("unchecked")
103          Map<String, String> temp = (Map<String, String>)copy;
104
105          nsMap = Collections.unmodifiableMap(temp);
106      }
107
108      /**
109       * Returns the XPath expression to be evaluated.
110       *
111       * @return the XPath expression to be evaluated
112       */
113      public String getXPath() {
114          return xpath;
115      }
116
117      /**
118       * Returns a map of namespace prefixes. Each key is a namespace prefix
119       * String that maps to a corresponding namespace URI
120       * String.
121       * <p>
122       * This implementation returns an {@link Collections#unmodifiableMap
123       * unmodifiable map}.
124       *
125       * @return a Map of namespace prefixes to namespace URIs (may

```

```

126     *   be empty, but never <code>null</code>)
127     */
128     @SuppressWarnings("rawtypes")
129     public Map getNamespaceMap() {
130         return nsMap;
131     }
132 }

```

### 13.9 XPathType.java

Secuencia 143: javax/xml/crypto/dsig/spec/XPathType.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: XPathType.java,v 1.4 2005/05/10 16:40:17 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.spec;
29
30 import java.util.Collections;
31 import java.util.Iterator;
32 import java.util.HashMap;
33 import java.util.Map;
34
35 /**
36 * The XML Schema Definition of the <code>XPath</code> element as defined in the
37 * <a href="http://www.w3.org/TR/xmlldsig-filter2">
38 * W3C Recommendation for XML-Signature XPath Filter 2.0</a>:
39 * <pre><code>
40 * <?xml schema xmlns="http://www.w3.org/2001/XMLSchema"
41 *     xmlns:xf="http://www.w3.org/2002/06/xmlldsig-filter2"
42 *     targetNamespace="http://www.w3.org/2002/06/xmlldsig-filter2"

```

```

43 *      version="0.1" elementFormDefault="qualified"&gt;
44 *
45 * <!--element name="XPath"
46 *      type="xf:XPathType"/&gt;
47 *
48 * <!--complexType name="XPathType"&gt;
49 * <!--simpleContent&gt;
50 * <!--extension base="string"&gt;
51 * <!--attribute name="Filter"&gt;
52 * <!--simpleType&gt;
53 * <!--restriction base="string"&gt;
54 * <!--enumeration value="intersect"/&gt;
55 * <!--enumeration value="subtract"/&gt;
56 * <!--enumeration value="union"/&gt;
57 * <!--/restriction&gt;
58 * <!--/simpleType&gt;
59 * <!--/attribute&gt;
60 * <!--/extension&gt;
61 * <!--/simpleContent&gt;
62 * <!--/complexType&gt;
63 * </code></pre>
64 *
65 * @author Sean Mullan
66 * @author JSR 105 Expert Group
67 * @since 1.6
68 * @see XPathFilter2ParameterSpec
69 */
70 public class XPathType {
71
72     /**
73      * Represents the filter set operation.
74      */
75     public static class Filter {
76         private final String operation;
77
78         private Filter(String operation) {
79             this.operation = operation;
80         }
81
82         /**
83          * Returns the string form of the operation.
84          *
85          * @return the string form of the operation
86          */
87         public String toString() {
88             return operation;
89         }
90
91         /**
92          * The intersect filter operation.
93          */
94         public static final Filter INTERSECT = new Filter("intersect");
95

```

```

96     /**
97      * The subtract filter operation.
98      */
99     public static final Filter SUBTRACT = new Filter("subtract");
100
101     /**
102      * The union filter operation.
103      */
104     public static final Filter UNION = new Filter("union");
105 }
106
107 private final String expression;
108 private final Filter filter;
109 private Map<String,String> nsMap;
110
111 /**
112  * Creates an <code>XPathType</code> instance with the specified XPath
113  * expression and filter.
114  *
115  * @param expression the XPath expression to be evaluated
116  * @param filter the filter operation ({@link Filter#INTERSECT},
117  *      {@link Filter#SUBTRACT}, or {@link Filter#UNION})
118  * @throws NullPointerException if <code>expression</code> or
119  *      <code>filter</code> is <code>null</code>
120  */
121 public XPathType(String expression, Filter filter) {
122     if (expression == null) {
123         throw new NullPointerException("expression cannot be null");
124     }
125     if (filter == null) {
126         throw new NullPointerException("filter cannot be null");
127     }
128     this.expression = expression;
129     this.filter = filter;
130     this.nsMap = Collections.emptyMap();
131 }
132
133 /**
134  * Creates an <code>XPathType</code> instance with the specified XPath
135  * expression, filter, and namespace map. The map is copied to protect
136  * against subsequent modification.
137  *
138  * @param expression the XPath expression to be evaluated
139  * @param filter the filter operation ({@link Filter#INTERSECT},
140  *      {@link Filter#SUBTRACT}, or {@link Filter#UNION})
141  * @param namespaceMap the map of namespace prefixes. Each key is a
142  *      namespace prefix <code>String</code> that maps to a corresponding
143  *      namespace URI <code>String</code>.
144  * @throws NullPointerException if <code>expression</code>,
145  *      <code>filter</code> or <code>namespaceMap</code> are
146  *      <code>null</code>
147  * @throws ClassCastException if any of the map's keys or entries are
148  *      not of type <code>String</code>

```

```

149     */
150     @SuppressWarnings("rawtypes")
151     public XPathType(String expression, Filter filter, Map namespaceMap) {
152         this(expression, filter);
153         if (namespaceMap == null) {
154             throw new NullPointerException("namespaceMap cannot be null");
155         }
156         Map<?,?> copy = new HashMap<>((Map<?,?>)namespaceMap);
157         Iterator<? extends Map.Entry<?,?>> entries = copy.entrySet().iterator();
158         while (entries.hasNext()) {
159             Map.Entry<?,?> me = entries.next();
160             if (!(me.getKey() instanceof String) ||
161                 !(me.getValue() instanceof String)) {
162                 throw new ClassCastException("not a String");
163             }
164         }
165
166         @SuppressWarnings("unchecked")
167         Map<String,String> temp = (Map<String,String>)copy;
168
169         nsMap = Collections.unmodifiableMap(temp);
170     }
171
172     /**
173      * Returns the XPath expression to be evaluated.
174      *
175      * @return the XPath expression to be evaluated
176      */
177     public String getExpression() {
178         return expression;
179     }
180
181     /**
182      * Returns the filter operation.
183      *
184      * @return the filter operation
185      */
186     public Filter getFilter() {
187         return filter;
188     }
189
190     /**
191      * Returns a map of namespace prefixes. Each key is a namespace prefix
192      * <code>String</code> that maps to a corresponding namespace URI
193      * <code>String</code>.
194      * <p>
195      * This implementation returns an {@link Collections#unmodifiableMap
196      * unmodifiable map}.
197      *
198      * @return a <code>Map</code> of namespace prefixes to namespace URIs
199      *      (may be empty, but never <code>null</code>)
200      */
201     @SuppressWarnings("rawtypes")

```

```

202     public Map getNamespaceMap() {
203         return nsMap;
204     }
205 }

```

### 13.10 XSLTTransformParameterSpec.java

Secuencia 144: javax/xml/crypto/dsig/spec/XSLTTransformParameterSpec.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25 /*
26 * $Id: XSLTTransformParameterSpec.java,v 1.4 2005/05/10 16:40:18 mullan Exp $
27 */
28 package javax.xml.crypto.dsig.spec;
29
30 import javax.xml.crypto.dsig.Transform;
31 import javax.xml.crypto.XMLStructure;
32
33 /**
34 * Parameters for the <a href="http://www.w3.org/TR/1999/REC-xslt-19991116">
35 * XSLT Transform Algorithm</a>.
36 * The parameters include a namespace-qualified stylesheet element.
37 *
38 * <p>An <code>XSLTTransformParameterSpec</code> is instantiated with a
39 * mechanism-dependent (ex: DOM) stylesheet element. For example:
40 * <pre>
41 *     DOMStructure stylesheet = new DOMStructure(element)
42 *     XSLTTransformParameterSpec spec = new XSLTransformParameterSpec(stylesheet);
43 * </pre>
44 * where <code>element</code> is an {@link org.w3c.dom.Element} containing
45 * the namespace-qualified stylesheet element.

```



```

46  *
47  * @author Sean Mullan
48  * @author JSR 105 Expert Group
49  * @since 1.6
50  * @see Transform
51  */
52  public final class XSLTTransformParameterSpec implements TransformParameterSpec{
53      private XMLStructure stylesheet;
54
55      /**
56       * Creates an <code>XSLTTransformParameterSpec</code> with the specified
57       * stylesheet.
58       *
59       * @param stylesheet the XSLT stylesheet to be used
60       * @throws NullPointerException if <code>stylesheet</code> is
61       *     <code>null</code>
62       */
63      public XSLTTransformParameterSpec(XMLStructure stylesheet) {
64          if (stylesheet == null) {
65              throw new NullPointerException();
66          }
67          this.stylesheet = stylesheet;
68      }
69
70      /**
71       * Returns the stylesheet.
72       *
73       * @return the stylesheet
74       */
75      public XMLStructure getStylesheet() {
76          return stylesheet;
77      }
78  }

```

## 14 Xml Datatype (javax.xml.datatype package)

### 14.1 DatatypeConfigurationException.java

Secuencia 145: javax/xml/datatype/DatatypeConfigurationException.java

```

1  /*
2  * Copyright (c) 2004, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```

```
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.datatype;
27
28  /**
29   * <p>Indicates a serious configuration error.</p>
30   *
31   * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
32   * @since 1.5
33   */
34
35  public class DatatypeConfigurationException extends Exception {
36
37      /**
38       * <p>Create a new <code>DatatypeConfigurationException</code> with
39       * no specified detail message and cause.</p>
40       */
41
42      public DatatypeConfigurationException() {
43          super();
44      }
45
46      /**
47       * <p>Create a new <code>DatatypeConfigurationException</code> with
48       * the specified detail message.</p>
49       *
50       * @param message The detail message.
51       */
52
53      public DatatypeConfigurationException(String message) {
54          super(message);
55      }
56
57      /**
58       * <p>Create a new <code>DatatypeConfigurationException</code> with
59       * the specified detail message and cause.</p>
60       *
61       * @param message The detail message.
62       * @param cause The cause. A <code>null</code> value is permitted, and indicates that the
63       * cause is nonexistent or unknown.
64       */
65
66      public DatatypeConfigurationException(String message, Throwable cause) {
67          super(message, cause);
68      }
69  }
```

```

67     }
68
69     /**
70      * <p>Create a new DatatypeConfigurationException with
71      * the specified cause.</p>
72      *
73      * @param cause The cause. A null value is permitted, and indicates that the
74      *           cause is nonexistent or unknown.
75      */
76     public DatatypeConfigurationException(Throwable cause) {
77         super(cause);
78     }
79 }

```

## 14.2 DatatypeConstants.java

Secuencia 146: javax/xml/datatype/DatatypeConstants.java

```

1  /*
2   * Copyright (c) 2004, 2006, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.datatype;
27
28 import javax.xml.XMLConstants;
29 import javax.xml.namespace.QName;
30
31 /**
32  * <p>Utility class to contain basic Datatype values as constants.</p>
33  *
34  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
35  * @since 1.5

```

```
36  */
37
38  public final class DatatypeConstants {
39
40      /**
41       * <p>Private constructor to prevent instantiation.</p>
42       */
43      private DatatypeConstants() {
44      }
45
46      /**
47       * Value for first month of year.
48       */
49      public static final int JANUARY = 1;
50
51      /**
52       * Value for second month of year.
53       */
54      public static final int FEBRUARY = 2;
55
56      /**
57       * Value for third month of year.
58       */
59      public static final int MARCH = 3;
60
61      /**
62       * Value for fourth month of year.
63       */
64      public static final int APRIL = 4;
65
66      /**
67       * Value for fifth month of year.
68       */
69      public static final int MAY = 5;
70
71      /**
72       * Value for sixth month of year.
73       */
74      public static final int JUNE = 6;
75
76      /**
77       * Value for seventh month of year.
78       */
79      public static final int JULY = 7;
80
81      /**
82       * Value for eighth month of year.
83       */
84      public static final int AUGUST = 8;
85
86      /**
87       * Value for ninth month of year.
88       */
```

```
89     public static final int SEPTEMBER = 9;
90
91     /**
92      * Value for tenth month of year.
93      */
94     public static final int OCTOBER = 10;
95
96     /**
97      * Value for eleven month of year.
98      */
99     public static final int NOVEMBER = 11;
100
101     /**
102      * Value for twelve month of year.
103      */
104     public static final int DECEMBER = 12;
105
106     /**
107      * <p>Comparison result.</p>
108      */
109     public static final int LESSER = -1;
110
111     /**
112      * <p>Comparison result.</p>
113      */
114     public static final int EQUAL = 0;
115
116     /**
117      * <p>Comparison result.</p>
118      */
119     public static final int GREATER = 1;
120
121     /**
122      * <p>Comparison result.</p>
123      */
124     public static final int INDETERMINATE = 2;
125
126     /**
127      * Designation that an "int" field is not set.
128      */
129     public static final int FIELD_UNDEFINED = Integer.MIN_VALUE;
130
131     /**
132      * <p>A constant that represents the years field.</p>
133      */
134     public static final Field YEARS = new Field("YEARS", 0);
135
136     /**
137      * <p>A constant that represents the months field.</p>
138      */
139     public static final Field MONTHS = new Field("MONTHS", 1);
140
141     /**
```

```

142     * <p>A constant that represents the days field.</p>
143     */
144     public static final Field DAYS = new Field("DAYS", 2);
145
146     /**
147     * <p>A constant that represents the hours field.</p>
148     */
149     public static final Field HOURS = new Field("HOURS", 3);
150
151     /**
152     * <p>A constant that represents the minutes field.</p>
153     */
154     public static final Field MINUTES = new Field("MINUTES", 4);
155
156     /**
157     * <p>A constant that represents the seconds field.</p>
158     */
159     public static final Field SECONDS = new Field("SECONDS", 5);
160
161     /**
162     * Type-safe enum class that represents six fields
163     * of the {@link Duration} class.
164     * @since 1.5
165     */
166     public static final class Field {
167
168         /**
169         * <p><code>String</code> representation of <code>Field</code>.</p>
170         */
171         private final String str;
172
173         /**
174         * <p>Unique id of the field.</p>
175         *
176         * <p>This value allows the {@link Duration} class to use switch
177         * statements to process fields.</p>
178         */
179         private final int id;
180
181         /**
182         * <p>Construct a <code>Field</code> with specified values.</p>
183         * @param str <code>String</code> representation of <code>Field</code>
184         * @param id <code>int</code> representation of <code>Field</code>
185         */
186         private Field(final String str, final int id) {
187             this.str = str;
188             this.id = id;
189         }
190
191         /**
192         * Returns a field name in English. This method
193         * is intended to be used for debugging/diagnosis
194         * and not for display to end-users.
195         *
196         * @return

```

```

195         *      a non-null valid String constant.
196         */
197         public String toString() { return str; }
198
199         /**
200         * <p>Get id of this Field.</p>
201         *
202         * @return Id of field.
203         */
204         public int getId() {
205             return id;
206         }
207     }
208
209     /**
210     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>dateTime</code>.</p>
211     */
212     public static final QName DATETIME = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "
        dateTime");
213
214     /**
215     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>time</code>.</p>
216     */
217     public static final QName TIME = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "time");
218
219     /**
220     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>date</code>.</p>
221     */
222     public static final QName DATE = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "date");
223
224     /**
225     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>gYearMonth</code>.</p>
226     */
227     public static final QName GYEARMONTH = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "
        gYearMonth");
228
229     /**
230     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>gMonthDay</code>.</p>
231     */
232     public static final QName GMONTHDAY = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "
        gMonthDay");
233
234     /**
235     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>gYear</code>.</p>
236     */
237     public static final QName GYEAR = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "gYear");
238
239     /**
240     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>gMonth</code>.</p>
241     */
242     public static final QName GMONTH = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "gMonth");
243
244     /**

```

```

245     * <p>Fully qualified name for W3C XML Schema 1.0 datatype <code>gDay</code>.</p>
246     */
247     public static final QName GDAY = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "gDay");
248
249     /**
250     * <p>Fully qualified name for W3C XML Schema datatype <code>duration</code>.</p>
251     */
252     public static final QName DURATION = new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, "
        duration");
253
254     /**
255     * <p>Fully qualified name for XQuery 1.0 and XPath 2.0 datatype <code>dayTimeDuration</
256     * code>.</p>
257     */
258     public static final QName DURATION_DAYTIME = new QName(XMLConstants.
259         W3C_XPATH_DATATYPE_NS_URI, "dayTimeDuration");
260
261     /**
262     * <p>Fully qualified name for XQuery 1.0 and XPath 2.0 datatype <code>yearMonthDuration</
263     * code>.</p>
264     */
265     public static final QName DURATION_YEARMONTH = new QName(XMLConstants.
266         W3C_XPATH_DATATYPE_NS_URI, "yearMonthDuration");
267
268     /**
269     * W3C XML Schema max timezone offset is -14:00. Zone offset is in minutes.
270     */
271     public static final int MAX_TIMEZONE_OFFSET = -14 * 60;
272
273     /**
274     * W3C XML Schema min timezone offset is +14:00. Zone offset is in minutes.
275     */
276     public static final int MIN_TIMEZONE_OFFSET = 14 * 60;
277 }

```

### 14.3 DatatypeFactory.java

Secuencia 147: javax/xml/datatype/DatatypeFactory.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```



```

15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.datatype;
27
28 import java.math.BigDecimal;
29 import java.math.BigInteger;
30 import java.util.GregorianCalendar;
31 import java.util.regex.Matcher;
32 import java.util.regex.Pattern;
33
34 /**
35  * <p>Factory that creates new <code>javax.xml.datatype</code> <code>Object</code>s that map XML to
    /from Java <code>Object</code>s.</p>
36  *
37  * <p>A new instance of the <code>DatatypeFactory</code> is created through the {@link #newInstance
    ()} method
38  * that uses the following implementation resolution mechanisms to determine an implementation:</p>
39  * <ol>
40  *   <li>
41  *     If the system property specified by {@link #DATATYPEFACTORY_PROPERTY}, "<code>javax.xml.
    datatype.DatatypeFactory</code>",
42  *     exists, a class with the name of the property value is instantiated.
43  *     Any Exception thrown during the instantiation process is wrapped as a {@link
    DatatypeConfigurationException}.
44  *   </li>
45  *   <li>
46  *     If the file ${JAVA_HOME}/lib/jaxp.properties exists, it is loaded in a {@link java.util.
    Properties} <code>Object</code>.
47  *     The <code>Properties</code> <code>Object</code> is then queried for the property as
    documented in the prior step
48  *     and processed as documented in the prior step.
49  *   </li>
50  *   <li>
51  *     Uses the service-provider loading facilities, defined by the {@link java.util.ServiceLoader}
    class, to attempt
52  *     to locate and load an implementation of the service using the {@linkplain
53  *     java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
54  *     the service-provider loading facility will use the {@linkplain
55  *     java.lang.Thread#getContextClassLoader() current thread's context class loader}
56  *     to attempt to load the service. If the context class
57  *     loader is null, the {@linkplain
58  *     ClassLoader#getSystemClassLoader() system class loader} will be used.
59  *   <br>
60  *   In case of {@link java.util.ServiceConfigurationError service

```

```

61 *      configuration error} a {@link javax.xml.datatype.DatatypeConfigurationException}
62 *      will be thrown.
63 *    </li>
64 *    <li>
65 *      The final mechanism is to attempt to instantiate the <code>Class</code> specified by
66 *      {@link #DATATYPEFACTORY_IMPLEMENTATION_CLASS}.
67 *      Any Exception thrown during the instantiation process is wrapped as a {@link
        DatatypeConfigurationException}.
68 *    </li>
69 * </ol>
70 *
71 * @author <a href="mailto:Joseph.Fialli@Sun.COM">Joseph Fialli</a>
72 * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
73 * @author <a href="mailto:Neeraj.Bajaj@sun.com">Neeraj Bajaj</a>
74 *
75 * @version $Revision: 1.13 $, $Date: 2010/03/11 23:10:53 $
76 * @since 1.5
77 */
78 public abstract class DatatypeFactory {
79
80     /**
81      * <p>Default property name as defined in JSR 206: Java(TM) API for XML Processing (JAXP)
        1.3.</p>
82      *
83      * <p>Default value is <code>javax.xml.datatype.DatatypeFactory</code>.</p>
84      */
85     public static final String DATATYPEFACTORY_PROPERTY =
86         // We use a String constant here, rather than calling
87         // DatatypeFactory.class.getName() - in order to make javadoc
88         // generate a See Also: Constant Field Value link.
89         "javax.xml.datatype.DatatypeFactory";
90
91     /**
92      * <p>Default implementation class name as defined in
93      * <em>JSR 206: Java(TM) API for XML Processing (JAXP) 1.3</em>.</p>
94      *
95      * <p>Implementers should specify the name of an appropriate class
96      * to be instantiated if no other implementation resolution mechanism
97      * succeeds.</p>
98      *
99      * <p>Users should not refer to this field; it is intended only to
100      * document a factory implementation detail.
101      * </p>
102      */
103     public static final String DATATYPEFACTORY_IMPLEMENTATION_CLASS =
104         // We use new String() here to prevent javadoc from generating
105         // a See Also: Constant Field Value link.
106         new String("com.sun.org.apache.xerces.internal.jaxp.datatype.DatatypeFactoryImpl");
107
108     /**
109      * http://www.w3.org/TR/xpath-datamodel/#xdtschema defines two regexps
110      * to constrain the value space of dayTimeDuration ([^YM]*[DT].*)
111      * and yearMonthDuration ([^DT]*). Note that these expressions rely on

```

```

112     * the fact that the value must be an xs:Duration, they simply exclude
113     * some Durations.
114     */
115     private static final Pattern XDTSHEMA_YMD =
116         Pattern.compile("[^DT]*");
117
118     private static final Pattern XDTSHEMA_DTD =
119         Pattern.compile("[^YM]*[DT].*");
120
121     /**
122     * <p>Protected constructor to prevent instantiation outside of package.</p>
123     *
124     * <p>Use {@link #newInstance()} to create a <code>DatatypeFactory</code>.</p>
125     */
126     protected DatatypeFactory() {
127     }
128
129     /**
130     * <p>Obtain a new instance of a <code>DatatypeFactory</code>.</p>
131     *
132     * <p>The implementation resolution mechanisms are <a href="#DatatypeFactory.newInstance">
133         defined</a> in this
134     * <code>Class</code>'s documentation.</p>
135     *
136     * @return New instance of a <code>DatatypeFactory</code>
137     *
138     * @throws DatatypeConfigurationException If the implementation is not
139     *     available or cannot be instantiated.
140     *
141     * @see #newInstance(String factoryClassName, ClassLoader classLoader)
142     */
143     public static DatatypeFactory newInstance()
144         throws DatatypeConfigurationException {
145
146         return FactoryFinder.find(
147             /* The default property name according to the JAXP spec */
148             DatatypeFactory.class,
149             /* The fallback implementation class name */
150             DATATYPEFACTORY_IMPLEMENTATION_CLASS);
151     }
152
153     /**
154     * <p>Obtain a new instance of a <code>DatatypeFactory</code> from class name.
155     * This function is useful when there are multiple providers in the classpath.
156     * It gives more control to the application as it can specify which provider
157     * should be loaded.</p>
158     *
159     * <p>Once an application has obtained a reference to a <code>DatatypeFactory</code>
160     * it can use the factory to configure and obtain datatype instances.</P>
161     *
162     * <h2>Tip for Trouble-shooting</h2>
163     * <p>Setting the <code>jaxp.debug</code> system property will cause

```

```

164 * this method to print a lot of debug messages
165 * to <code>System.err</code> about what it is doing and where it is looking at.</p>
166 *
167 * <p> If you have problems try:</p>
168 * <pre>
169 * java -Djaxp.debug=1 YourProgram ....
170 * </pre>
171 *
172 * @param factoryClassName fully qualified factory class name that provides implementation of <
173 *   code>javax.xml.datatype.DatatypeFactory</code>.
174 * @param classLoader <code>ClassLoader</code> used to load the factory class. If <code>null</code>/
175 *   code>
176 *   current <code>Thread</code>'s context classLoader is used to load the
177 *   factory class.
178 * @return New instance of a <code>DatatypeFactory</code>
179 * @throws DatatypeConfigurationException if <code>factoryClassName</code> is <code>null</code>
180 *   >, or
181 *   the factory class cannot be loaded, instantiated.
182 * @see #newInstance()
183 *
184 * @since 1.6
185 */
186 public static DatatypeFactory newInstance(String factoryClassName, ClassLoader classLoader)
187     throws DatatypeConfigurationException {
188     return FactoryFinder.newInstance(DatatypeFactory.class,
189         factoryClassName, classLoader, false);
190 }
191
192 /**
193 * <p>Obtain a new instance of a <code>Duration</code>
194 * specifying the <code>Duration</code> as its string representation, "PnYnMnDnHnMnS",
195 * as defined in XML Schema 1.0 section 3.2.6.1.</p>
196 *
197 * <p>XML Schema Part 2: Datatypes, 3.2.6 duration, defines <code>duration</code> as:</p>
198 * <blockquote>
199 * duration represents a duration of time.
200 * The value space of duration is a six-dimensional space where the coordinates designate the
201 * Gregorian year, month, day, hour, minute, and second components defined in Section 5.5.3.2
202 * of [ISO 8601], respectively.
203 * These components are ordered in their significance by their order of appearance i.e. as
204 * year, month, day, hour, minute, and second.
205 * </blockquote>
206 * <p>All six values are set and available from the created {@link Duration}</p>
207 *
208 * <p>The XML Schema specification states that values can be of an arbitrary size.
209 * Implementations may chose not to or be incapable of supporting arbitrarily large and/or
210 * small values.
211 * An {@link UnsupportedOperationException} will be thrown with a message indicating
212 * implementation limits

```

```

210     * if implementation capacities are exceeded.</p>
211     *
212     * @param lexicalRepresentation <code>String</code> representation of a <code>Duration</code>.
213     *
214     * @return New <code>Duration</code> created from parsing the <code>lexicalRepresentation</code>
215     *     >.
216     *
217     * @throws IllegalArgumentException If <code>lexicalRepresentation</code> is not a valid
218     *     representation of a <code>Duration</code>.
219     * @throws UnsupportedOperationException If implementation cannot support requested values.
220     * @throws NullPointerException if <code>lexicalRepresentation</code> is <code>null</code>.
221     */
222     public abstract Duration newDuration(final String lexicalRepresentation);
223
224     /**
225     * <p>Obtain a new instance of a <code>Duration</code>
226     * specifying the <code>Duration</code> as milliseconds.</p>
227     *
228     * <p>XML Schema Part 2: Datatypes, 3.2.6 duration, defines <code>duration</code> as:</p>
229     * <blockquote>
230     * duration represents a duration of time.
231     * The value space of duration is a six-dimensional space where the coordinates designate the
232     * Gregorian year, month, day, hour, minute, and second components defined in Section 5.5.3.2
233     * of [ISO 8601], respectively.
234     * These components are ordered in their significance by their order of appearance i.e. as
235     * year, month, day, hour, minute, and second.
236     * </blockquote>
237     * <p>All six values are set by computing their values from the specified milliseconds
238     * and are available using the <code>get</code> methods of the created {@link Duration}.
239     * The values conform to and are defined by:</p>
240     * <ul>
241     * <li>ISO 8601:2000(E) Section 5.5.3.2 Alternative format</li>
242     * <li><a href="http://www.w3.org/TR/xmlschema-2/#isoformats">
243     * W3C XML Schema 1.0 Part 2, Appendix D, ISO 8601 Date and Time Formats</a>
244     * </li>
245     * <li>{@link XMLGregorianCalendar} Date/Time Datatype Field Mapping Between XML Schema 1.0
246     * and Java Representation</li>
247     * </ul>
248     *
249     * <p>The default start instance is defined by {@link GregorianCalendar}'s use of the start of
250     * the epoch: i.e.,
251     *
252     * {@link java.util.Calendar#YEAR} = 1970,
253     * {@link java.util.Calendar#MONTH} = {@link java.util.Calendar#JANUARY},
254     * {@link java.util.Calendar#DATE} = 1, etc.
255     * This is important as there are variations in the Gregorian Calendar,
256     * e.g. leap years have different days in the month = {@link java.util.Calendar#FEBRUARY}
257     * so the result of {@link Duration#getMonths()} and {@link Duration#getDays()} can be
258     * influenced.</p>
259     *
260     * @param durationInMilliSeconds Duration in milliseconds to create.
261     *
262     * @return New <code>Duration</code> representing <code>durationInMilliSeconds</code>.
263     */

```

```

257 public abstract Duration newDuration(final long durationInMilliseconds);
258
259 /**
260  * <p>Obtain a new instance of a <code>Duration</code>
261  * specifying the <code>Duration</code> as isPositive, years, months, days, hours, minutes,
262  * seconds.</p>
263  *
264  * <p>The XML Schema specification states that values can be of an arbitrary size.
265  * Implementations may chose not to or be incapable of supporting arbitrarily large and/or
266  * small values.
267  *
268  * An {@link UnsupportedOperationException} will be thrown with a message indicating
269  * implementation limits
270  * if implementation capacities are exceeded.</p>
271  *
272  * <p>A <code>null</code> value indicates that field is not set.</p>
273  *
274  * @param isPositive Set to <code>>false</code> to create a negative duration. When the length
275  * of the duration is zero, this parameter will be ignored.
276  * @param years of this <code>Duration</code>
277  * @param months of this <code>Duration</code>
278  * @param days of this <code>Duration</code>
279  * @param hours of this <code>Duration</code>
280  * @param minutes of this <code>Duration</code>
281  * @param seconds of this <code>Duration</code>
282  *
283  * @return New <code>Duration</code> created from the specified values.
284  *
285  * @throws IllegalArgumentException If the values are not a valid representation of a
286  * <code>Duration</code>: if all the fields (years, months, ...) are null or
287  * if any of the fields is negative.
288  * @throws UnsupportedOperationException If implementation cannot support requested values.
289  */
290 public abstract Duration newDuration(
291     final boolean isPositive,
292     final BigInteger years,
293     final BigInteger months,
294     final BigInteger days,
295     final BigInteger hours,
296     final BigInteger minutes,
297     final BigDecimal seconds);
298
299 /**
300  * <p>Obtain a new instance of a <code>Duration</code>
301  * specifying the <code>Duration</code> as isPositive, years, months, days, hours, minutes,
302  * seconds.</p>
303  *
304  * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>
305  *
306  * @param isPositive Set to <code>>false</code> to create a negative duration. When the length
307  * of the duration is zero, this parameter will be ignored.
308  * @param years of this <code>Duration</code>
309  * @param months of this <code>Duration</code>
310  * @param days of this <code>Duration</code>

```

```
306 * @param hours of this <code>Duration</code>
307 * @param minutes of this <code>Duration</code>
308 * @param seconds of this <code>Duration</code>
309 *
310 * @return New <code>Duration</code> created from the specified values.
311 *
312 * @throws IllegalArgumentException If the values are not a valid representation of a
313 * <code>Duration</code>: if any of the fields is negative.
314 *
315 * @see #newDuration(
316 *     boolean isPositive,
317 *     BigInteger years,
318 *     BigInteger months,
319 *     BigInteger days,
320 *     BigInteger hours,
321 *     BigInteger minutes,
322 *     BigDecimal seconds)
323 */
324 public Duration newDuration(
325     final boolean isPositive,
326     final int years,
327     final int months,
328     final int days,
329     final int hours,
330     final int minutes,
331     final int seconds) {
332
333     // years may not be set
334     BigInteger realYears = (years != DatatypeConstants.FIELD_UNDEFINED) ? BigInteger.
        valueOf((long) years) : null;
335
336     // months may not be set
337     BigInteger realMonths = (months != DatatypeConstants.FIELD_UNDEFINED) ? BigInteger.
        valueOf((long) months) : null;
338
339     // days may not be set
340     BigInteger realDays = (days != DatatypeConstants.FIELD_UNDEFINED) ? BigInteger.valueOf
        ((long) days) : null;
341
342     // hours may not be set
343     BigInteger realHours = (hours != DatatypeConstants.FIELD_UNDEFINED) ? BigInteger.
        valueOf((long) hours) : null;
344
345     // minutes may not be set
346     BigInteger realMinutes = (minutes != DatatypeConstants.FIELD_UNDEFINED) ? BigInteger.
        valueOf((long) minutes) : null;
347
348     // seconds may not be set
349     BigDecimal realSeconds = (seconds != DatatypeConstants.FIELD_UNDEFINED) ? BigDecimal.
        valueOf((long) seconds) : null;
350
351     return newDuration(
352         isPositive,
```

```

353         realYears,
354         realMonths,
355         realDays,
356         realHours,
357         realMinutes,
358         realSeconds
359     );
360 }
361
362 /**
363  * <p>Create a Duration of type xdt:dayTimeDuration by parsing its <
364  *   code>String</code> representation,
365  *   "<em>PnDTnHnMnS</em>", <a href="http://www.w3.org/TR/xpath-datamodel#dayTimeDuration">
366  *   XQuery 1.0 and XPath 2.0 Data Model, xdt:dayTimeDuration</a>.</p>
367  *   <p>The datatype xdt:dayTimeDuration is a subtype of xs:duration
368  *   whose lexical representation contains only day, hour, minute, and second components.
369  *   This datatype resides in the namespace http://www.w3.org/2003/11/xpath-datatypes</code>
370  *   </p>
371  *   <p>All four values are set and available from the created {<@link Duration>}</p>
372  *   <p>The XML Schema specification states that values can be of an arbitrary size.
373  *   Implementations may chose not to or be incapable of supporting arbitrarily large and/or
374  *   small values.
375  *   An {<@link UnsupportedOperationException>} will be thrown with a message indicating
376  *   implementation limits
377  *   if implementation capacities are exceeded.</p>
378  *   @param lexicalRepresentation Lexical representation of a duration.
379  *   @return New Duration created using the specified lexicalRepresentation</code>
380  *   </code>.
381  *   @throws IllegalArgumentException If lexicalRepresentation</code> is not a valid
382  *   representation of a Duration</code> expressed only in terms of days and time.
383  *   @throws UnsupportedOperationException If implementation cannot support requested values.
384  *   @throws NullPointerException If lexicalRepresentation</code> is null</code>.
385  */
386 public Duration newDurationDayTime(final String lexicalRepresentation) {
387     // lexicalRepresentation must be non-null
388     if (lexicalRepresentation == null) {
389         throw new NullPointerException(
390             "Trying to create an xdt:dayTimeDuration with an invalid"
391             + " lexical representation of \"null\"");
392     }
393
394     // test lexicalRepresentation against spec regex
395     Matcher matcher = XDTSHEMA_DTD.matcher(lexicalRepresentation);
396     if (!matcher.matches()) {
397         throw new IllegalArgumentException(
398             "Trying to create an xdt:dayTimeDuration with an invalid"
399             + " lexical representation of \"" + lexicalRepresentation

```



```

400         + "\", data model requires years and months only.");
401     }
402
403     return newDuration(lexicalRepresentation);
404 }
405
406 /**
407  * <p>Create a <code>Duration</code> of type <code>xdt:dayTimeDuration</code> using the
408     specified milliseconds as defined in
409  * <a href="http://www.w3.org/TR/xpath-datamodel#dayTimeDuration">
410  *   XQuery 1.0 and XPath 2.0 Data Model, xdt:dayTimeDuration</a>.</p>
411  *
412  * <p>The datatype <code>xdt:dayTimeDuration</code> is a subtype of <code>xs:duration</code>
413  * whose lexical representation contains only day, hour, minute, and second components.
414  * This datatype resides in the namespace <code>http://www.w3.org/2003/11/xpath-datatypes</code>
415  * </p>
416  *
417  * <p>All four values are set by computing their values from the specified milliseconds
418  * and are available using the <code>get</code> methods of the created {@link Duration}.
419  * The values conform to and are defined by:</p>
420  * <ul>
421  * <li>ISO 8601:2000(E) Section 5.5.3.2 Alternative format</li>
422  * <li><a href="http://www.w3.org/TR/xmlschema-2/#isoformats">
423  *   W3C XML Schema 1.0 Part 2, Appendix D, ISO 8601 Date and Time Formats</a>
424  * </li>
425  * <li>{@link XMLGregorianCalendar} Date/Time Datatype Field Mapping Between XML Schema 1.0
426  *   and Java Representation</li>
427  * </ul>
428  *
429  * <p>The default start instance is defined by {@link GregorianCalendar}'s use of the start of
430  * the epoch: i.e.,
431  * {@link java.util.Calendar#YEAR} = 1970,
432  * {@link java.util.Calendar#MONTH} = {@link java.util.Calendar#JANUARY},
433  * {@link java.util.Calendar#DATE} = 1, etc.
434  * This is important as there are variations in the Gregorian Calendar,
435  * e.g. leap years have different days in the month = {@link java.util.Calendar#FEBRUARY}
436  * so the result of {@link Duration#getDays()} can be influenced.</p>
437  *
438  * <p>Any remaining milliseconds after determining the day, hour, minute and second are
439  * discarded.</p>
440  *
441  * @param durationInMilliseconds Milliseconds of <code>Duration</code> to create.
442  *
443  * @return New <code>Duration</code> created with the specified <code>durationInMilliseconds</code>
444  *
445  * @see <a href="http://www.w3.org/TR/xpath-datamodel#dayTimeDuration">
446  *   XQuery 1.0 and XPath 2.0 Data Model, xdt:dayTimeDuration</a>
447  */
448 public Duration newDurationDayTime(final long durationInMilliseconds) {
449
450     return newDuration(durationInMilliseconds);
451 }

```

```

447
448 /**
449  * <p>Create a Duration of type xdt:dayTimeDuration using the
      specified
450  * day, hour, minute and second as defined
      in
451  * <a href="http://www.w3.org/TR/xpath-datamodel#dayTimeDuration">
452  *   XQuery 1.0 and XPath 2.0 Data Model, xdt:dayTimeDuration</a>.</p>
453  *
454  * <p>The datatype xdt:dayTimeDuration is a subtype of xs:duration
455  * whose lexical representation contains only day, hour, minute, and second components.
456  * This datatype resides in the namespace http://www.w3.org/2003/11/xpath-datatypes
457  * </p>
458  *
459  * <p>The XML Schema specification states that values can be of an arbitrary size.
      Implementations may chose not to or be incapable of supporting arbitrarily large and/or
      small values.
460  * An {@link UnsupportedOperationException} will be thrown with a message indicating
      implementation limits
461  * if implementation capacities are exceeded.</p>
462  *
463  * <p>A null value indicates that field is not set.</p>
464  *
465  * @param isPositive Set to false to create a negative duration. When the length
466  *   of the duration is zero, this parameter will be ignored.
467  * @param day Day of Duration.
468  * @param hour Hour of Duration.
469  * @param minute Minute of Duration.
470  * @param second Second of Duration.
471  *
472  * @return New Duration created with the specified day, hour
473  *   code, minute
474  *   code, and second.
475  *
476  * @throws IllegalArgumentException If the values are not a valid representation of a
477  *   Duration: if all the fields (day, hour, ...) are null or
478  *   if any of the fields is negative.
479  * @throws UnsupportedOperationException If implementation cannot support requested values.
480  */
481 public Duration newDurationDayTime(
482     final boolean isPositive,
483     final BigInteger day,
484     final BigInteger hour,
485     final BigInteger minute,
486     final BigInteger second) {
487
488     return newDuration(
489         isPositive,
490         null, // years
491         null, // months
492         day,
493         hour,
494         minute,

```

```

494         (second != null)? new BigDecimal(second):null
495     );
496 }
497
498 /**
499  * <p>Create a Duration of type xdt:dayTimeDuration using the
      specified
500  * day, hour, minute and second as defined
      in
501  * <a href="http://www.w3.org/TR/xpath-datamodel#dayTimeDuration">
502  *   XQuery 1.0 and XPath 2.0 Data Model, xdt:dayTimeDuration</a>.</p>
503  *
504  * <p>The datatype xdt:dayTimeDuration is a subtype of xs:duration
505  * whose lexical representation contains only day, hour, minute, and second components.
506  * This datatype resides in the namespace http://www.w3.org/2003/11/xpath-datatypes
507  * </p>
508  * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>
509  *
510  * @param isPositive Set to false to create a negative duration. When the length
511  *   of the duration is zero, this parameter will be ignored.
512  * @param day Day of Duration.
513  * @param hour Hour of Duration.
514  * @param minute Minute of Duration.
515  * @param second Second of Duration.
516  *
517  * @return New Duration created with the specified day, hour/
518  *   code, minute/
519  *   code, and second/
520  *   code.
521  *
522  * @throws IllegalArgumentException If the values are not a valid representation of a
523  *   Duration: if any of the fields (day, hour, ...) is negative.
524  */
525 public Duration newDurationDayTime(
526     final boolean isPositive,
527     final int day,
528     final int hour,
529     final int minute,
530     final int second) {
531
532     return newDurationDayTime(
533         isPositive,
534         BigInteger.valueOf((long) day),
535         BigInteger.valueOf((long) hour),
536         BigInteger.valueOf((long) minute),
537         BigInteger.valueOf((long) second)
538     );
539 }
540
541 /**
542  * <p>Create a Duration of type xdt:yearMonthDuration by parsing its
543  *   String representation,
544  *   "em>PnYnM</em>", <a href="http://www.w3.org/TR/xpath-datamodel#yearMonthDuration">

```

```

542 *   XQuery 1.0 and XPath 2.0 Data Model, xdt:yearMonthDuration</a>.</p>
543 *
544 *   <p>The datatype <code>xdt:yearMonthDuration</code> is a subtype of <code>xs:duration</code>
545 *   whose lexical representation contains only year and month components.
546 *   This datatype resides in the namespace {@link javax.xml.XMLConstants#
       W3C_XPATH_DATATYPE_NS_URI}</p>
547 *
548 *   <p>Both values are set and available from the created {@link Duration}</p>
549 *
550 *   <p>The XML Schema specification states that values can be of an arbitrary size.
551 *   Implementations may chose not to or be incapable of supporting arbitrarily large and/or
       small values.
552 *   An {@link UnsupportedOperationException} will be thrown with a message indicating
       implementation limits
553 *   if implementation capacities are exceeded.</p>
554 *
555 *   @param lexicalRepresentation Lexical representation of a duration.
556 *
557 *   @return New <code>Duration</code> created using the specified <code>lexicalRepresentation</
       code>.
558 *
559 *   @throws IllegalArgumentException If <code>lexicalRepresentation</code> is not a valid
       representation of a <code>Duration</code> expressed only in terms of years and months.
560 *   @throws UnsupportedOperationException If implementation cannot support requested values.
561 *   @throws NullPointerException If <code>lexicalRepresentation</code> is <code>null</code>.
562 */
563 public Duration newDurationYearMonth(
564     final String lexicalRepresentation) {
565
566     // lexicalRepresentation must be non-null
567     if (lexicalRepresentation == null) {
568         throw new NullPointerException(
569             "Trying to create an xdt:yearMonthDuration with an invalid"
570             + " lexical representation of \"null\"");
571     }
572
573     // test lexicalRepresentation against spec regex
574     Matcher matcher = XDTSHEMA_YMD.matcher(lexicalRepresentation);
575     if (!matcher.matches()) {
576         throw new IllegalArgumentException(
577             "Trying to create an xdt:yearMonthDuration with an invalid"
578             + " lexical representation of \"" + lexicalRepresentation
579             + "\", data model requires days and times only.");
580     }
581
582     return newDuration(lexicalRepresentation);
583 }
584
585 /**
586 *   <p>Create a <code>Duration</code> of type <code>xdt:yearMonthDuration</code> using the
       specified milliseconds as defined in
587 *   <a href="http://www.w3.org/TR/xpath-datamodel#yearMonthDuration">
588 *   XQuery 1.0 and XPath 2.0 Data Model, xdt:yearMonthDuration</a>.</p>

```

```

589      *
590      * <p>The datatype <code>xdt:yearMonthDuration</code> is a subtype of <code>xs:duration</code>
591      * whose lexical representation contains only year and month components.
592      * This datatype resides in the namespace {@link javax.xml.XMLConstants#
        W3C_XPATH_DATATYPE_NS_URI}.</p>
593      *
594      * <p>Both values are set by computing their values from the specified milliseconds
595      * and are available using the <code>get</code> methods of the created {@link Duration}.
596      * The values conform to and are defined by:</p>
597      * <ul>
598      * <li>ISO 8601:2000(E) Section 5.5.3.2 Alternative format</li>
599      * <li><a href="http://www.w3.org/TR/xmlschema-2/#isoformats">
600      *   W3C XML Schema 1.0 Part 2, Appendix D, ISO 8601 Date and Time Formats</a>
601      * </li>
602      * <li>{@link XMLGregorianCalendar} Date/Time Datatype Field Mapping Between XML Schema 1.0
        and Java Representation</li>
603      * </ul>
604      *
605      * <p>The default start instance is defined by {@link GregorianCalendar}'s use of the start of
        the epoch: i.e.,
606      * {@link java.util.Calendar#YEAR} = 1970,
607      * {@link java.util.Calendar#MONTH} = {@link java.util.Calendar#JANUARY},
608      * {@link java.util.Calendar#DATE} = 1, etc.
609      * This is important as there are variations in the Gregorian Calendar,
610      * e.g. leap years have different days in the month = {@link java.util.Calendar#FEBRUARY}
611      * so the result of {@link Duration#getMonths()} can be influenced.</p>
612      *
613      * <p>Any remaining milliseconds after determining the year and month are discarded.</p>
614      *
615      * @param durationInMilliseconds Milliseconds of <code>Duration</code> to create.
616      *
617      * @return New <code>Duration</code> created using the specified <code>durationInMilliseconds</code>.
618      */
619      public Duration newDurationYearMonth(
620          final long durationInMilliseconds) {
621
622          // create a Duration that only has sign, year & month
623          // Duration is immutable, so need to create a new Duration
624          // implementations may override this method in a more efficient way
625          Duration fullDuration = newDuration(durationInMilliseconds);
626          boolean isPositive = (fullDuration.getSign() == -1) ? false : true;
627          BigInteger years =
628              (BigInteger) fullDuration.getField(DatatypeConstants.YEARS);
629          if (years == null) { years = BigInteger.ZERO; }
630          BigInteger months =
631              (BigInteger) fullDuration.getField(DatatypeConstants.MONTHS);
632          if (months == null) { months = BigInteger.ZERO; }
633
634          return newDurationYearMonth(isPositive, years, months);
635      }
636
637      /**

```

```

638 * <p>Create a Duration of type xdt:yearMonthDuration using the
        specified
639 * year and month as defined in
640 * <a href="http://www.w3.org/TR/xpath-datamodel#yearMonthDuration">
641 *   XQuery 1.0 and XPath 2.0 Data Model, xdt:yearMonthDuration</a>.</p>
642 *
643 * <p>The XML Schema specification states that values can be of an arbitrary size.
644 * Implementations may chose not to or be incapable of supporting arbitrarily large and/or
        small values.
645 * An {@link UnsupportedOperationException} will be thrown with a message indicating
        implementation limits
646 * if implementation capacities are exceeded.</p>
647 *
648 * <p>A null value indicates that field is not set.</p>
649 *
650 * @param isPositive Set to false to create a negative duration. When the length
651 *   of the duration is zero, this parameter will be ignored.
652 * @param year Year of Duration.
653 * @param month Month of Duration.
654 *
655 * @return New Duration created using the specified year and month.
        month</code>.
656 *
657 * @throws IllegalArgumentException If the values are not a valid representation of a
658 *   Duration: if all of the fields (year, month) are null or
659 *   if any of the fields is negative.
660 * @throws UnsupportedOperationException If implementation cannot support requested values.
661 */
662 public Duration newDurationYearMonth(
663     final boolean isPositive,
664     final BigInteger year,
665     final BigInteger month) {
666
667     return newDuration(
668         isPositive,
669         year,
670         month,
671         null, // days
672         null, // hours
673         null, // minutes
674         null // seconds
675     );
676 }
677
678 /**
679 * <p>Create a Duration of type xdt:yearMonthDuration using the
        specified
680 * year and month as defined in
681 * <a href="http://www.w3.org/TR/xpath-datamodel#yearMonthDuration">
682 *   XQuery 1.0 and XPath 2.0 Data Model, xdt:yearMonthDuration</a>.</p>
683 *
684 * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>
685 *

```

```

686 * @param isPositive Set to false to create a negative duration. When the length
687 *   of the duration is zero, this parameter will be ignored.
688 * @param year Year of Duration.
689 * @param month Month of Duration.
690 *
691 * @return New Duration created using the specified year and month.
692 *
693 * @throws IllegalArgumentException If the values are not a valid representation of a
694 *   Duration: if any of the fields (year, month) is negative.
695 */
696 public Duration newDurationYearMonth(
697     final boolean isPositive,
698     final int year,
699     final int month) {
700
701     return newDurationYearMonth(
702         isPositive,
703         BigInteger.valueOf((long) year),
704         BigInteger.valueOf((long) month));
705 }
706
707 /**
708 * <p>Create a new instance of an XMLGregorianCalendar.</p>
709 *
710 * <p>All date/time datatype fields set to {@link DatatypeConstants#FIELD_UNDEFINED} or null.</p>
711 *
712 * @return New XMLGregorianCalendar with all date/time datatype fields set to
713 *   {@link DatatypeConstants#FIELD_UNDEFINED} or null.
714 */
715 public abstract XMLGregorianCalendar newXMLGregorianCalendar();
716
717 /**
718 * <p>Create a new XMLGregorianCalendar by parsing the String as a lexical representation.</p>
719 *
720 * <p>Parsing the lexical string representation is defined in
721 *   <a href="http://www.w3.org/TR/xmlschema-2/#dateTime-order">XML Schema 1.0 Part 2, Section
722 *     3.2. [7-14].1,
723 *   <em>Lexical Representation</em>.</a></p>
724 *
725 * <p>The string representation may not have any leading and trailing whitespaces.</p>
726 *
727 * <p>The parsing is done field by field so that
728 *   the following holds for any lexically correct String x:</p>
729 * <pre>
730 *   newXMLGregorianCalendar(x).toXMLFormat().equals(x)
731 * </pre>
732 * <p>Except for the noted lexical/canonical representation mismatches
733 *   listed in <a href="http://www.w3.org/2001/05/xmlschema-errata#e2-45">
734 *   XML Schema 1.0 errata, Section 3.2.7.2</a>.</p>
735 *
736 * @param lexicalRepresentation Lexical representation of one the eight XML Schema date/time

```



```

    datatypes.
736  *
737  * @return <code>XMLGregorianCalendar</code> created from the <code>lexicalRepresentation</code>
    >.
738  *
739  * @throws IllegalArgumentException If the <code>lexicalRepresentation</code> is not a valid <
    code>XMLGregorianCalendar</code>.
740  * @throws NullPointerException If <code>lexicalRepresentation</code> is <code>null</code>.
741  */
742  public abstract XMLGregorianCalendar newXMLGregorianCalendar(final String lexicalRepresentation
    );
743
744  /**
745  * <p>Create an <code>XMLGregorianCalendar</code> from a {@link GregorianCalendar}.</p>
746  *
747  * <table border="2" rules="all" cellpadding="2">
748  *   <thead>
749  *     <tr>
750  *       <th align="center" colspan="2">
751  *         Field by Field Conversion from
752  *         {@link GregorianCalendar} to an {@link XMLGregorianCalendar}
753  *       </th>
754  *     </tr>
755  *     <tr>
756  *       <th><code>java.util.GregorianCalendar</code> field</th>
757  *       <th><code>javax.xml.datatype.XMLGregorianCalendar</code> field</th>
758  *     </tr>
759  *   </thead>
760  *   <tbody>
761  *     <tr>
762  *       <td><code>ERA == GregorianCalendar.BC ? -YEAR : YEAR</code></td>
763  *       <td>{@link XMLGregorianCalendar#setYear(int year)}</td>
764  *     </tr>
765  *     <tr>
766  *       <td><code>MONTH + 1</code></td>
767  *       <td>{@link XMLGregorianCalendar#setMonth(int month)}</td>
768  *     </tr>
769  *     <tr>
770  *       <td><code>DAY_OF_MONTH</code></td>
771  *       <td>{@link XMLGregorianCalendar#setDay(int day)}</td>
772  *     </tr>
773  *     <tr>
774  *       <td><code>HOUR_OF_DAY, MINUTE, SECOND, MILLISECOND</code></td>
775  *       <td>{@link XMLGregorianCalendar#setTime(int hour, int minute, int second, BigDecimal
    fractional)}</td>
776  *     </tr>
777  *     <tr>
778  *       <td>
779  *         <code>(ZONE_OFFSET + DST_OFFSET) / (60*1000)</code><br/>
780  *         <em>(in minutes)</em>
781  *       </td>
782  *       <td>{@link XMLGregorianCalendar#setTimezone(int offset)}<sup><em>*</em></sup>
783  *     </td>

```



```

784 *     </tr>
785 *     </tbody>
786 * </table>
787 * <p><em>*</em>conversion loss of information. It is not possible to represent
788 * a <code>java.util.GregorianCalendar</code> daylight savings timezone id in the
789 * XML Schema 1.0 date/time datatype representation.</p>
790 *
791 * <p>To compute the return value's <code>TimeZone</code> field,
792 * <ul>
793 * <li>when <code>this.getTimezone() != FIELD_UNDEFINED</code>,
794 * create a <code>java.util.TimeZone</code> with a custom timezone id
795 * using the <code>this.getTimezone()</code>.</li>
796 * <li>else use the <code>GregorianCalendar</code> default timezone value
797 * for the host is defined as specified by
798 * <code>java.util.TimeZone.getDefault()</code>.</li></ul>
799 *
800 * @param cal <code>java.util.GregorianCalendar</code> used to create <code>
      XMLGregorianCalendar</code>
801 *
802 * @return <code>XMLGregorianCalendar</code> created from <code>java.util.GregorianCalendar</
      code>
803 *
804 * @throws NullPointerException If <code>cal</code> is <code>null</code>.
805 */
806 public abstract XMLGregorianCalendar newXMLGregorianCalendar(final GregorianCalendar cal);
807
808 /**
809 * <p>Constructor allowing for complete value spaces allowed by
810 * W3C XML Schema 1.0 recommendation for xsd:dateTime and related
811 * builtin datatypes. Note that <code>year</code> parameter supports
812 * arbitrarily large numbers and fractionalSecond has infinite
813 * precision.</p>
814 *
815 * <p>A <code>null</code> value indicates that field is not set.</p>
816 *
817 * @param year of <code>XMLGregorianCalendar</code> to be created.
818 * @param month of <code>XMLGregorianCalendar</code> to be created.
819 * @param day of <code>XMLGregorianCalendar</code> to be created.
820 * @param hour of <code>XMLGregorianCalendar</code> to be created.
821 * @param minute of <code>XMLGregorianCalendar</code> to be created.
822 * @param second of <code>XMLGregorianCalendar</code> to be created.
823 * @param fractionalSecond of <code>XMLGregorianCalendar</code> to be created.
824 * @param timezone of <code>XMLGregorianCalendar</code> to be created.
825 *
826 * @return <code>XMLGregorianCalendar</code> created from specified values.
827 *
828 * @throws IllegalArgumentException If any individual parameter's value is outside the maximum
      value constraint for the field
829 * as determined by the Date/Time Data Mapping table in {@link XMLGregorianCalendar}
830 * or if the composite values constitute an invalid <code>XMLGregorianCalendar</code>
      instance
831 * as determined by {@link XMLGregorianCalendar#isValid()}.
832 */

```

```

833 public abstract XMLGregorianCalendar newXMLGregorianCalendar(
834     final BigInteger year,
835     final int month,
836     final int day,
837     final int hour,
838     final int minute,
839     final int second,
840     final BigDecimal fractionalSecond,
841     final int timezone);
842
843 /**
844  * <p>Constructor of value spaces that a
845  * <code>java.util.GregorianCalendar</code> instance would need to convert to an
846  * <code>XMLGregorianCalendar</code> instance.</p>
847  *
848  * <p><code>XMLGregorianCalendar eon</code> and
849  * <code>fractionalSecond</code> are set to <code>null</code></p>
850  *
851  * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>
852  *
853  * @param year of <code>XMLGregorianCalendar</code> to be created.
854  * @param month of <code>XMLGregorianCalendar</code> to be created.
855  * @param day of <code>XMLGregorianCalendar</code> to be created.
856  * @param hour of <code>XMLGregorianCalendar</code> to be created.
857  * @param minute of <code>XMLGregorianCalendar</code> to be created.
858  * @param second of <code>XMLGregorianCalendar</code> to be created.
859  * @param millisecond of <code>XMLGregorianCalendar</code> to be created.
860  * @param timezone of <code>XMLGregorianCalendar</code> to be created.
861  *
862  * @return <code>XMLGregorianCalendar</code> created from specified values.
863  *
864  * @throws IllegalArgumentException If any individual parameter's value is outside the maximum
865  *         value constraint for the field
866  *         as determined by the Date/Time Data Mapping table in {@link XMLGregorianCalendar}
867  *         or if the composite values constitute an invalid <code>XMLGregorianCalendar</code>
868  *         instance
869  *         as determined by {@link XMLGregorianCalendar#isValid()}.
870  */
871 public XMLGregorianCalendar newXMLGregorianCalendar(
872     final int year,
873     final int month,
874     final int day,
875     final int hour,
876     final int minute,
877     final int second,
878     final int millisecond,
879     final int timezone) {
880
881     // year may be undefined
882     BigInteger realYear = (year != DatatypeConstants.FIELD_UNDEFINED) ? BigInteger.valueOf
        ((long) year) : null;
883
884     // millisecond may be undefined

```

```

883         // millisecond must be >= 0 millisecond <= 1000
884         BigDecimal realMillisecond = null; // undefined value
885         if (millisecond != DatatypeConstants.FIELD_UNDEFINED) {
886             if (millisecond < 0 || millisecond > 1000) {
887                 throw new IllegalArgumentException(
888                     "javax.xml.datatype.DatatypeFactory#
                        newXMLGregorianCalendar("
889                     + "int year, int month, int day, int hour, int
                        minute, int second, int millisecond, int
                        timezone)"
890                     + "with invalid millisecond: " + millisecond
891                     );
892             }
893
894             realMillisecond = BigDecimal.valueOf((long) millisecond).movePointLeft(3);
895         }
896
897         return newXMLGregorianCalendar(
898             realYear,
899             month,
900             day,
901             hour,
902             minute,
903             second,
904             realMillisecond,
905             timezone
906         );
907     }
908
909     /**
910      * <p>Create a Java representation of XML Schema builtin datatype <code>date</code> or <code>g
      * </code>.</p>
911      *
912      * <p>For example, an instance of <code>gYear</code> can be created invoking this factory
913      * with <code>month</code> and <code>day</code> parameters set to
914      * {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
915      *
916      * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>
917      *
918      * @param year of <code>XMLGregorianCalendar</code> to be created.
919      * @param month of <code>XMLGregorianCalendar</code> to be created.
920      * @param day of <code>XMLGregorianCalendar</code> to be created.
921      * @param timezone offset in minutes. {@link DatatypeConstants#FIELD_UNDEFINED} indicates
      * optional field is not set.
922      *
923      * @return <code>XMLGregorianCalendar</code> created from parameter values.
924      *
925      * @see DatatypeConstants#FIELD_UNDEFINED
926      *
927      * @throws IllegalArgumentException If any individual parameter's value is outside the maximum
      * value constraint for the field
928      * as determined by the Date/Time Data Mapping table in {@link XMLGregorianCalendar}
929      * or if the composite values constitute an invalid <code>XMLGregorianCalendar</code>

```

```

    instance
930     * as determined by {@link XMLGregorianCalendar#isValid()}.
931     */
932     public XMLGregorianCalendar newXMLGregorianCalendarDate(
933         final int year,
934         final int month,
935         final int day,
936         final int timezone) {
937
938         return newXMLGregorianCalendar(
939             year,
940             month,
941             day,
942             DatatypeConstants.FIELD_UNDEFINED, // hour
943             DatatypeConstants.FIELD_UNDEFINED, // minute
944             DatatypeConstants.FIELD_UNDEFINED, // second
945             DatatypeConstants.FIELD_UNDEFINED, // millisecond
946             timezone);
947     }
948
949     /**
950     * <p>Create a Java instance of XML Schema builtin datatype <code>time</code>.</p>
951     *
952     * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>
953     *
954     * @param hours number of hours
955     * @param minutes number of minutes
956     * @param seconds number of seconds
957     * @param timezone offset in minutes. {@link DatatypeConstants#FIELD_UNDEFINED} indicates
958         optional field is not set.
959     *
960     * @return <code>XMLGregorianCalendar</code> created from parameter values.
961     *
962     * @throws IllegalArgumentException If any individual parameter's value is outside the maximum
963         value constraint for the field
964     * as determined by the Date/Time Data Mapping table in {@link XMLGregorianCalendar}
965     * or if the composite values constitute an invalid <code>XMLGregorianCalendar</code>
966     instance
967     * as determined by {@link XMLGregorianCalendar#isValid()}.
968     *
969     * @see DatatypeConstants#FIELD_UNDEFINED
970     */
971     public XMLGregorianCalendar newXMLGregorianCalendarTime(
972         final int hours,
973         final int minutes,
974         final int seconds,
975         final int timezone) {
976
977         return newXMLGregorianCalendar(
978             DatatypeConstants.FIELD_UNDEFINED, // Year
979             DatatypeConstants.FIELD_UNDEFINED, // Month
980             DatatypeConstants.FIELD_UNDEFINED, // Day
981             hours,

```

```

979         minutes,
980         seconds,
981         DatatypeConstants.FIELD_UNDEFINED, //Millisecond
982         timezone);
983     }
984
985     /**
986     * <p>Create a Java instance of XML Schema builtin datatype time.</p>
987     *
988     * <p>A <code>null</code> value indicates that field is not set.</p>
989     * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>
990     *
991     * @param hours number of hours
992     * @param minutes number of minutes
993     * @param seconds number of seconds
994     * @param fractionalSecond value of <code>null</code> indicates that this optional field is not
995       set.
996     * @param timezone offset in minutes. {@link DatatypeConstants#FIELD_UNDEFINED} indicates
997       optional field is not set.
998     *
999     * @return <code>XMLGregorianCalendar</code> created from parameter values.
1000    *
1001    * @see DatatypeConstants#FIELD_UNDEFINED
1002    *
1003    * @throws IllegalArgumentException If any individual parameter's value is outside the maximum
1004      value constraint for the field
1005    * as determined by the Date/Time Data Mapping table in {@link XMLGregorianCalendar}
1006    * or if the composite values constitute an invalid <code>XMLGregorianCalendar</code>
1007    * instance
1008    * as determined by {@link XMLGregorianCalendar#isValid()}.
1009    */
1010    public XMLGregorianCalendar newXMLGregorianCalendarTime(
1011        final int hours,
1012        final int minutes,
1013        final int seconds,
1014        final BigDecimal fractionalSecond,
1015        final int timezone) {
1016
1017        return newXMLGregorianCalendar(
1018            null, // year
1019            DatatypeConstants.FIELD_UNDEFINED, // month
1020            DatatypeConstants.FIELD_UNDEFINED, // day
1021            hours,
1022            minutes,
1023            seconds,
1024            fractionalSecond,
1025            timezone);
1026    }
1027
1028    /**
1029    * <p>Create a Java instance of XML Schema builtin datatype time.</p>
1030    *
1031    * <p>A {@link DatatypeConstants#FIELD_UNDEFINED} value indicates that field is not set.</p>

```

```

1028 *
1029 * @param hours number of hours
1030 * @param minutes number of minutes
1031 * @param seconds number of seconds
1032 * @param milliseconds number of milliseconds
1033 * @param timezone offset in minutes. {@link DatatypeConstants#FIELD_UNDEFINED} indicates
    optional field is not set.
1034 *
1035 * @return <code>XMLGregorianCalendar</code> created from parameter values.
1036 *
1037 * @see DatatypeConstants#FIELD_UNDEFINED
1038 *
1039 * @throws IllegalArgumentException If any individual parameter's value is outside the maximum
    value constraint for the field
1040 * as determined by the Date/Time Data Mapping table in {@link XMLGregorianCalendar}
1041 * or if the composite values constitute an invalid <code>XMLGregorianCalendar</code>
    instance
1042 * as determined by {@link XMLGregorianCalendar#isValid()}.
1043 */
1044 public XMLGregorianCalendar newXMLGregorianCalendarTime(
1045     final int hours,
1046     final int minutes,
1047     final int seconds,
1048     final int milliseconds,
1049     final int timezone) {
1050
1051     // millisecond may be undefined
1052     // millisecond must be >= 0 millisecond <= 1000
1053     BigDecimal realMilliseconds = null; // undefined value
1054     if (milliseconds != DatatypeConstants.FIELD_UNDEFINED) {
1055         if (milliseconds < 0 || milliseconds > 1000) {
1056             throw new IllegalArgumentException(
1057                 "javax.xml.datatype.DatatypeFactory#
                    newXMLGregorianCalendarTime("
1058                 + "int hours, int minutes, int seconds, int
                    milliseconds, int timezone)"
1059                 + "with invalid milliseconds: " + milliseconds
1060                 );
1061         }
1062
1063         realMilliseconds = BigDecimal.valueOf((long) milliseconds).movePointLeft(3);
1064     }
1065
1066     return newXMLGregorianCalendarTime(
1067         hours,
1068         minutes,
1069         seconds,
1070         realMilliseconds,
1071         timezone
1072     );
1073 }
1074 }

```

## 14.4 Duration.java

Secuencia 148: javax/xml/datatype/Duration.java

```
1  /*
2  * Copyright (c) 2003, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.datatype;
27
28 import java.math.BigDecimal;
29 import java.math.BigInteger;
30 import java.util.Calendar;
31 import java.util.Date;
32 import java.util.GregorianCalendar;
33
34 import javax.xml.namespace.QName;
35
36 /**
37 * <p>Immutable representation of a time span as defined in
38 * the W3C XML Schema 1.0 specification.</p>
39 *
40 * <p>A Duration object represents a period of Gregorian time,
41 * which consists of six fields (years, months, days, hours,
42 * minutes, and seconds) plus a sign (+/-) field.</p>
43 *
44 * <p>The first five fields have non-negative (>=0) integers or null
45 * (which represents that the field is not set),
46 * and the seconds field has a non-negative decimal or null.
47 * A negative sign indicates a negative duration.</p>
48 *
49 * <p>This class provides a number of methods that make it easy
50 * to use for the duration datatype of XML Schema 1.0 with
```

```

51 * the errata.</p>
52 *
53 * <h2>Order relationship</h2>
54 * <p>Duration objects only have partial order, where two values A and B
55 * maybe either:</p>
56 * <ol>
57 * <li>A<B (A is shorter than B)
58 * <li>A>B (A is longer than B)
59 * <li>A==B (A and B are of the same duration)
60 * <li>A<B (Comparison between A and B is indeterminate)
61 * </ol>
62 *
63 * <p>For example, 30 days cannot be meaningfully compared to one month.
64 * The {@link #compare(Duration duration)} method implements this
65 * relationship.</p>
66 *
67 * <p>See the {@link #isLongerThan(Duration)} method for details about
68 * the order relationship among Duration objects.</p>
69 *
70 * <h2>Operations over Duration</h2>
71 * <p>This class provides a set of basic arithmetic operations, such
72 * as addition, subtraction and multiplication.
73 * Because durations don't have total order, an operation could
74 * fail for some combinations of operations. For example, you cannot
75 * subtract 15 days from 1 month. See the javadoc of those methods
76 * for detailed conditions where this could happen.</p>
77 *
78 * <p>Also, division of a duration by a number is not provided because
79 * the Duration class can only deal with finite precision
80 * decimal numbers. For example, one cannot represent 1 sec divided by 3.</p>
81 *
82 * <p>However, you could substitute a division by 3 with multiplying
83 * by numbers such as 0.3 or 0.333.</p>
84 *
85 * <h2>Range of allowed values</h2>
86 * <p>
87 * Because some operations of Duration rely on {@link Calendar}
88 * even though {@link Duration} can hold very large or very small values,
89 * some of the methods may not work correctly on such Durations.
90 * The impacted methods document their dependency on {@link Calendar}.
91 *
92 * @author <a href="mailto:Joseph.Fialli@Sun.COM">Joseph Fialli</a>
93 * @author <a href="mailto:Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
94 * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
95 * @author <a href="mailto:Sunitha.Reddy@Sun.com">Sunitha Reddy</a>
96 * @see XMLGregorianCalendar#add(Duration)
97 * @since 1.5
98 */
99 public abstract class Duration {
100
101     /**
102      * <p>Debugging true or false.</p>
103      */

```



```

104 private static final boolean DEBUG = true;
105
106 /**
107  * Default no-arg constructor.
108  *
109  * <p>Note: Always use the {@link DatatypeFactory} to
110  * construct an instance of <code>Duration</code>.
111  * The constructor on this class cannot be guaranteed to
112  * produce an object with a consistent state and may be
113  * removed in the future.</p>
114  */
115 public Duration() {
116 }
117
118 /**
119  * <p>Return the name of the XML Schema date/time type that this instance
120  * maps to. Type is computed based on fields that are set,
121  * i.e. {@link #isSet(DatatypeConstants.Field field)} == <code>true</code>.</p>
122  *
123  * <table border="2" rules="all" cellpadding="2">
124  *   <thead>
125  *     <tr>
126  *       <th align="center" colspan="7">
127  *         Required fields for XML Schema 1.0 Date/Time Datatypes.<br/>
128  *         <i>(timezone is optional for all date/time datatypes)</i>
129  *       </th>
130  *     </tr>
131  *   </thead>
132  *   <tbody>
133  *     <tr>
134  *       <td>Datatype</td>
135  *       <td>year</td>
136  *       <td>month</td>
137  *       <td>day</td>
138  *       <td>hour</td>
139  *       <td>minute</td>
140  *       <td>second</td>
141  *     </tr>
142  *     <tr>
143  *       <td>{@link DatatypeConstants#DURATION}</td>
144  *       <td>X</td>
145  *       <td>X</td>
146  *       <td>X</td>
147  *       <td>X</td>
148  *       <td>X</td>
149  *       <td>X</td>
150  *     </tr>
151  *     <tr>
152  *       <td>{@link DatatypeConstants#DURATION_DAYTIME}</td>
153  *       <td></td>
154  *       <td></td>
155  *       <td>X</td>
156  *       <td>X</td>

```

```

157 *      <td>X</td>
158 *      <td>X</td>
159 *    </tr>
160 *    <tr>
161 *      <td>{@link DatatypeConstants#DURATION_YEARMONTH}</td>
162 *      <td>X</td>
163 *      <td>X</td>
164 *      <td></td>
165 *      <td></td>
166 *      <td></td>
167 *      <td></td>
168 *    </tr>
169 *  </tbody>
170 * </table>
171 *
172 * @return one of the following constants:
173 *   {@link DatatypeConstants#DURATION},
174 *   {@link DatatypeConstants#DURATION_DAYTIME} or
175 *   {@link DatatypeConstants#DURATION_YEARMONTH}.
176 *
177 * @throws IllegalStateException If the combination of set fields does not match one of the XML
178 *   Schema date/time datatypes.
179 */
180 public QName getXMLSchemaType() {
181     boolean yearSet = isSet(DatatypeConstants.YEARS);
182     boolean monthSet = isSet(DatatypeConstants.MONTHS);
183     boolean daySet = isSet(DatatypeConstants.DAYS);
184     boolean hourSet = isSet(DatatypeConstants.HOURS);
185     boolean minuteSet = isSet(DatatypeConstants.MINUTES);
186     boolean secondSet = isSet(DatatypeConstants.SECONDS);
187
188     // DURATION
189     if (yearSet
190         && monthSet
191         && daySet
192         && hourSet
193         && minuteSet
194         && secondSet) {
195         return DatatypeConstants.DURATION;
196     }
197
198     // DURATION_DAYTIME
199     if (!yearSet
200         && !monthSet
201         && daySet
202         && hourSet
203         && minuteSet
204         && secondSet) {
205         return DatatypeConstants.DURATION_DAYTIME;
206     }
207
208     // DURATION_YEARMONTH

```

```

209     if (yearSet
210         && monthSet
211         && !daySet
212         && !hourSet
213         && !minuteSet
214         && !secondSet) {
215         return DatatypeConstants.DURATION_YEARMONTH;
216     }
217
218     // nothing matches
219     throw new IllegalStateException(
220         "javax.xml.datatype.Duration#getXMLSchemaType():"
221         + " this Duration does not match one of the XML Schema date/time datatypes:"
222         + " year set = " + yearSet
223         + " month set = " + monthSet
224         + " day set = " + daySet
225         + " hour set = " + hourSet
226         + " minute set = " + minuteSet
227         + " second set = " + secondSet
228     );
229 }
230
231 /**
232  * Returns the sign of this duration in -1,0, or 1.
233  *
234  * @return
235  *     -1 if this duration is negative, 0 if the duration is zero,
236  *     and 1 if the duration is positive.
237  */
238 public abstract int getSign();
239
240 /**
241  * <p>Get the years value of this <code>Duration</code> as an <code>int</code> or <code>0</code>
242  *   > if not present.</p>
243  *
244  * <p><code>getYears()</code> is a convenience method for
245  *   {@link #getField\(DatatypeConstants.Field field\) getField\(DatatypeConstants.YEARS\)}.</p>
246  *
247  * <p>As the return value is an <code>int</code>, an incorrect value will be returned for <code>
248  *   >Duration</code>s
249  *   with years that go beyond the range of an <code>int</code>.
250  *   Use {@link #getField\(DatatypeConstants.Field field\) getField\(DatatypeConstants.YEARS\)} to
251  *   avoid possible loss of precision.</p>
252  *
253  * @return If the years field is present, return its value as an <code>int</code>, else return
254  *   <code>0</code>.
255  */
256 public int getYears() {
257     return getField(DatatypeConstants.YEARS).intValue();
258 }
259
260 /**
261  * Obtains the value of the MONTHS field as an integer value,

```

```
258     * or 0 if not present.
259     *
260     * This method works just like {@link #getYears()} except
261     * that this method works on the MONTHS field.
262     *
263     * @return Months of this <code>Duration</code>.
264     */
265     public int getMonths() {
266         return getField(DatatypeConstants.MONTHS).intValue();
267     }
268
269     /**
270     * Obtains the value of the DAYS field as an integer value,
271     * or 0 if not present.
272     *
273     * This method works just like {@link #getYears()} except
274     * that this method works on the DAYS field.
275     *
276     * @return Days of this <code>Duration</code>.
277     */
278     public int getDays() {
279         return getField(DatatypeConstants.DAYS).intValue();
280     }
281
282     /**
283     * Obtains the value of the HOURS field as an integer value,
284     * or 0 if not present.
285     *
286     * This method works just like {@link #getYears()} except
287     * that this method works on the HOURS field.
288     *
289     * @return Hours of this <code>Duration</code>.
290     *
291     */
292     public int getHours() {
293         return getField(DatatypeConstants.HOURS).intValue();
294     }
295
296     /**
297     * Obtains the value of the MINUTES field as an integer value,
298     * or 0 if not present.
299     *
300     * This method works just like {@link #getYears()} except
301     * that this method works on the MINUTES field.
302     *
303     * @return Minutes of this <code>Duration</code>.
304     *
305     */
306     public int getMinutes() {
307         return getField(DatatypeConstants.MINUTES).intValue();
308     }
309
310     /**
```

```

311     * Obtains the value of the SECONDS field as an integer value,
312     * or 0 if not present.
313     *
314     * This method works just like {@link #getYears()} except
315     * that this method works on the SECONDS field.
316     *
317     * @return seconds in the integer value. The fraction of seconds
318     *     will be discarded (for example, if the actual value is 2.5,
319     *     this method returns 2)
320     */
321     public int getSeconds() {
322         return getField(DatatypeConstants.SECONDS).intValue();
323     }
324
325     /**
326     * <p>Returns the length of the duration in milli-seconds.</p>
327     *
328     * <p>If the seconds field carries more digits than milli-second order,
329     * those will be simply discarded (or in other words, rounded to zero.)
330     * For example, for any Calendar value <code>x</code>,</p>
331     * <pre>
332     * <code>new Duration("PT10.00099S").getTimeInMills(x) == 10000</code>.
333     * <code>new Duration("-PT10.00099S").getTimeInMills(x) == -10000</code>.
334     * </pre>
335     *
336     * <p>
337     * Note that this method uses the {@link #addTo(Calendar)} method,
338     * which may work incorrectly with <code>Duration</code> objects with
339     * very large values in its fields. See the {@link #addTo(Calendar)}
340     * method for details.
341     *
342     * @param startInstant
343     *     The length of a month/year varies. The <code>startInstant</code> is
344     *     used to disambiguate this variance. Specifically, this method
345     *     returns the difference between <code>startInstant</code> and
346     *     <code>startInstant+duration</code>
347     *
348     * @return milliseconds between <code>startInstant</code> and
349     *     <code>startInstant</code> plus this <code>Duration</code>
350     *
351     * @throws NullPointerException if <code>startInstant</code> parameter
352     *     is null.
353     *
354     */
355     public long getTimeInMills(final Calendar startInstant) {
356         Calendar cal = (Calendar) startInstant.clone();
357         addTo(cal);
358         return getCalendarTimeInMills(cal)
359             - getCalendarTimeInMills(startInstant);
360     }
361
362     /**
363     * <p>Returns the length of the duration in milli-seconds.</p>

```

```

364 *
365 * <p>If the seconds field carries more digits than milli-second order,
366 * those will be simply discarded (or in other words, rounded to zero.)
367 * For example, for any <code>Date</code> value <code>x</code>,</p>
368 * <pre>
369 * <code>new Duration("PT10.00099S").getTimeInMills(x) == 10000</code>.
370 * <code>new Duration("-PT10.00099S").getTimeInMills(x) == -10000</code>.
371 * </pre>
372 *
373 * <p>
374 * Note that this method uses the {@link #addTo(Date)} method,
375 * which may work incorrectly with <code>Duration</code> objects with
376 * very large values in its fields. See the {@link #addTo(Date)}
377 * method for details.
378 *
379 * @param startInstant
380 *     The length of a month/year varies. The <code>startInstant</code> is
381 *     used to disambiguate this variance. Specifically, this method
382 *     returns the difference between <code>startInstant</code> and
383 *     <code>startInstant+duration</code>.
384 *
385 * @throws NullPointerException
386 *     If the startInstant parameter is null.
387 *
388 * @return milliseconds between <code>startInstant</code> and
389 *     <code>startInstant</code> plus this <code>Duration</code>
390 *
391 * @see #getTimeInMillis(Calendar)
392 */
393 public long getTimeInMillis(final Date startInstant) {
394     Calendar cal = new GregorianCalendar();
395     cal.setTime(startInstant);
396     this.addTo(cal);
397     return getCalendarTimeInMillis(cal) - startInstant.getTime();
398 }
399
400 /**
401 * Gets the value of a field.
402 *
403 * Fields of a duration object may contain arbitrary large value.
404 * Therefore this method is designed to return a {@link Number} object.
405 *
406 * In case of YEARS, MONTHS, DAYS, HOURS, and MINUTES, the returned
407 * number will be a non-negative integer. In case of seconds,
408 * the returned number may be a non-negative decimal value.
409 *
410 * @param field
411 *     one of the six Field constants (YEARS,MONTHS,DAYS,HOURS,
412 *     MINUTES, or SECONDS.)
413 * @return
414 *     If the specified field is present, this method returns
415 *     a non-null non-negative {@link Number} object that
416 *     represents its value. If it is not present, return null.

```

```

417 * For YEARS, MONTHS, DAYS, HOURS, and MINUTES, this method
418 * returns a {@link java.math.BigInteger} object. For SECONDS, this
419 * method returns a {@link java.math.BigDecimal}.
420 *
421 * @throws NullPointerException If the <code>field</code> is <code>>null</code>.
422 */
423 public abstract Number getField(final DatatypeConstants.Field field);
424
425 /**
426 * Checks if a field is set.
427 *
428 * A field of a duration object may or may not be present.
429 * This method can be used to test if a field is present.
430 *
431 * @param field
432 *     one of the six Field constants (YEARS,MONTHS,DAYS,HOURS,
433 *     MINUTES, or SECONDS.)
434 * @return
435 *     true if the field is present. false if not.
436 *
437 * @throws NullPointerException
438 *     If the field parameter is null.
439 */
440 public abstract boolean isSet(final DatatypeConstants.Field field);
441
442 /**
443 * <p>Computes a new duration whose value is <code>this+rhs</code>.</p>
444 *
445 * <p>For example,</p>
446 * <pre>
447 * "1 day" + "-3 days" = "-2 days"
448 * "1 year" + "1 day" = "1 year and 1 day"
449 * "-(1 hour,50 minutes)" + "-20 minutes" = "-(1 hours,70 minutes)"
450 * "15 hours" + "-3 days" = "-(2 days,9 hours)"
451 * "1 year" + "-1 day" = IllegalStateException
452 * </pre>
453 *
454 * <p>Since there's no way to meaningfully subtract 1 day from 1 month,
455 * there are cases where the operation fails in
456 * {@link IllegalStateException}.</p>
457 *
458 * <p>
459 * Formally, the computation is defined as follows.</p>
460 * <p>
461 * Firstly, we can assume that two <code>Duration</code>s to be added
462 * are both positive without losing generality (i.e.,
463 * <code>(-X)+Y=Y-X</code>, <code>X+(-Y)=X-Y</code>,
464 * <code>(-X)+(-Y)=-(X+Y)</code>)
465 *
466 * <p>
467 * Addition of two positive <code>Duration</code>s are simply defined as
468 * field by field addition where missing fields are treated as 0.
469 * <p>

```

```

470 * A field of the resulting Duration will be unset if and
471 * only if respective fields of two input Durations are unset.
472 * <p>
473 * Note that lhs.add(rhs) will be always successful if
474 * lhs.signum()*rhs.signum() != -1 or both of them are
475 * normalized.</p>
476 *
477 * @param rhs Duration to add to this Duration
478 *
479 * @return
480 *     non-null valid Duration object.
481 *
482 * @throws NullPointerException
483 *     If the rhs parameter is null.
484 * @throws IllegalStateException
485 *     If two durations cannot be meaningfully added. For
486 *     example, adding negative one day to one month causes
487 *     this exception.
488 *
489 *
490 * @see #subtract(Duration)
491 */
492 public abstract Duration add(final Duration rhs);
493
494 /**
495 * Adds this duration to a {@link Calendar} object.
496 *
497 * <p>
498 * Calls {@link java.util.Calendar#add(int,int)} in the
499 * order of YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, and MILLISECONDS
500 * if those fields are present. Because the {@link Calendar} class
501 * uses int to hold values, there are cases where this method
502 * won't work correctly (for example if values of fields
503 * exceed the range of int.)
504 * </p>
505 *
506 * <p>
507 * Also, since this duration class is a Gregorian duration, this
508 * method will not work correctly if the given {@link Calendar}
509 * object is based on some other calendar systems.
510 * </p>
511 *
512 * <p>
513 * Any fractional parts of this Duration object
514 * beyond milliseconds will be simply ignored. For example, if
515 * this duration is "P1.23456S", then 1 is added to SECONDS,
516 * 234 is added to MILLISECONDS, and the rest will be unused.
517 * </p>
518 *
519 * <p>
520 * Note that because {@link Calendar#add(int, int)} is using
521 * int, Duration with values beyond the
522 * range of int in its fields

```



```

523     * will cause overflow/underflow to the given {@link Calendar}.
524     * {@link XMLGregorianCalendar#add(Duration)} provides the same
525     * basic operation as this method while avoiding
526     * the overflow/underflow issues.
527     *
528     * @param calendar
529     *     A calendar object whose value will be modified.
530     * @throws NullPointerException
531     *     if the calendar parameter is null.
532     */
533     public abstract void addTo(Calendar calendar);
534
535     /**
536     * Adds this duration to a {@link Date} object.
537     *
538     * <p>
539     * The given date is first converted into
540     * a {@link java.util.GregorianCalendar}, then the duration
541     * is added exactly like the {@link #addTo(Calendar)} method.
542     *
543     * <p>
544     * The updated time instant is then converted back into a
545     * {@link Date} object and used to update the given {@link Date} object.
546     *
547     * <p>
548     * This somewhat redundant computation is necessary to unambiguously
549     * determine the duration of months and years.
550     *
551     * @param date
552     *     A date object whose value will be modified.
553     * @throws NullPointerException
554     *     if the date parameter is null.
555     */
556     public void addTo(Date date) {
557
558         // check data parameter
559         if (date == null) {
560             throw new NullPointerException(
561                 "Cannot call "
562                 + this.getClass().getName()
563                 + "#addTo(Date date) with date == null."
564             );
565         }
566
567         Calendar cal = new GregorianCalendar();
568         cal.setTime(date);
569         this.addTo(cal);
570         date.setTime(getCalendarTimeInMillis(cal));
571     }
572
573     /**
574     * <p>Computes a new duration whose value is <code>this-rhs</code>.</p>
575     *

```

```

576 * <p>For example:</p>
577 * <pre>
578 * "1 day" - "-3 days" = "4 days"
579 * "1 year" - "1 day" = IllegalStateException
580 * "-(1 hour,50 minutes)" - "-20 minutes" = "-(1hours,30 minutes)"
581 * "15 hours" - "-3 days" = "3 days and 15 hours"
582 * "1 year" - "-1 day" = "1 year and 1 day"
583 * </pre>
584 *
585 * <p>Since there's no way to meaningfully subtract 1 day from 1 month,
586 * there are cases where the operation fails in {@link IllegalStateException}.</p>
587 *
588 * <p>Formally the computation is defined as follows.
589 * First, we can assume that two <code>Duration</code>s are both positive
590 * without losing generality. (i.e.,
591 * <code>(-X)-Y=-(X+Y)</code>, <code>X-(-Y)=X+Y</code>,
592 * <code>(-X)-(-Y)=-(X-Y)</code>)</p>
593 *
594 * <p>Then two durations are subtracted field by field.
595 * If the sign of any non-zero field <code>F</code> is different from
596 * the sign of the most significant field,
597 * 1 (if <code>F</code> is negative) or -1 (otherwise)
598 * will be borrowed from the next bigger unit of <code>F</code>.</p>
599 *
600 * <p>This process is repeated until all the non-zero fields have
601 * the same sign.</p>
602 *
603 * <p>If a borrow occurs in the days field (in other words, if
604 * the computation needs to borrow 1 or -1 month to compensate
605 * days), then the computation fails by throwing an
606 * {@link IllegalStateException}.</p>
607 *
608 * @param rhs <code>Duration</code> to subtract from this <code>Duration</code>.
609 *
610 * @return New <code>Duration</code> created from subtracting <code>rhs</code> from this <code>
        Duration</code>.
611 *
612 * @throws IllegalStateException
613 *     If two durations cannot be meaningfully subtracted. For
614 *     example, subtracting one day from one month causes
615 *     this exception.
616 *
617 * @throws NullPointerException
618 *     If the rhs parameter is null.
619 *
620 * @see #add(Duration)
621 */
622 public Duration subtract(final Duration rhs) {
623     return add(rhs.negate());
624 }
625
626 /**
627 * <p>Computes a new duration whose value is <code>factor</code> times

```

```

628     * longer than the value of this duration.</p>
629     *
630     * <p>This method is provided for the convenience.
631     * It is functionally equivalent to the following code:</p>
632     * <pre>
633     * multiply(new BigDecimal(String.valueOf(factor)))
634     * </pre>
635     *
636     * @param factor Factor times longer of new <code>Duration</code> to create.
637     *
638     * @return New <code>Duration</code> that is <code>factor</code>times longer than this <code>
        Duration</code>.
639     *
640     * @see #multiply(BigDecimal)
641     */
642     public Duration multiply(int factor) {
643         return multiply(new BigDecimal(String.valueOf(factor)));
644     }
645
646     /**
647     * Computes a new duration whose value is <code>factor</code> times
648     * longer than the value of this duration.
649     *
650     * <p>
651     * For example,
652     * <pre>
653     * "P1M" (1 month) * "12" = "P12M" (12 months)
654     * "PT1M" (1 min) * "0.3" = "PT18S" (18 seconds)
655     * "P1M" (1 month) * "1.5" = IllegalStateException
656     * </pre>
657     *
658     * <p>
659     * Since the <code>Duration</code> class is immutable, this method
660     * doesn't change the value of this object. It simply computes
661     * a new Duration object and returns it.
662     *
663     * <p>
664     * The operation will be performed field by field with the precision
665     * of {@link BigDecimal}. Since all the fields except seconds are
666     * restricted to hold integers,
667     * any fraction produced by the computation will be
668     * carried down toward the next lower unit. For example,
669     * if you multiply "P1D" (1 day) with "0.5", then it will be 0.5 day,
670     * which will be carried down to "PT12H" (12 hours).
671     * When fractions of month cannot be meaningfully carried down
672     * to days, or year to months, this will cause an
673     * {@link IllegalStateException} to be thrown.
674     * For example if you multiple one month by 0.5.</p>
675     *
676     * <p>
677     * To avoid {@link IllegalStateException}, use
678     * the {@link #normalizeWith(Calendar)} method to remove the years
679     * and months fields.

```

```

680      *
681      * @param factor to multiply by
682      *
683      * @return
684      *     returns a non-null valid Duration object
685      *
686      * @throws IllegalStateException if operation produces fraction in
687      * the months field.
688      *
689      * @throws NullPointerException if the factor parameter is
690      * null.
691      *
692      */
693 public abstract Duration multiply(final BigDecimal factor);
694
695 /**
696  * Returns a new Duration object whose
697  * value is -this.
698  *
699  * <p>
700  * Since the Duration class is immutable, this method
701  * doesn't change the value of this object. It simply computes
702  * a new Duration object and returns it.
703  *
704  * @return
705  *     always return a non-null valid Duration object.
706  */
707 public abstract Duration negate();
708
709 /**
710  * <p>Converts the years and months fields into the days field
711  * by using a specific time instant as the reference point.</p>
712  *
713  * <p>For example, duration of one month normalizes to 31 days
714  * given the start time instance "July 8th 2003, 17:40:32".</p>
715  *
716  * <p>Formally, the computation is done as follows:</p>
717  * <ol>
718  * <li>the given Calendar object is cloned</li>
719  * <li>the years, months and days fields will be added to the {@link Calendar} object
720  *     by using the {@link Calendar#add(int,int)} method</li>
721  * <li>the difference between the two Calendars is computed in milliseconds and converted to
722  *     days,
723  *     if a remainder occurs due to Daylight Savings Time, it is discarded</li>
724  * <li>the computed days, along with the hours, minutes and seconds
725  *     fields of this duration object is used to construct a new
726  *     Duration object.</li>
727  * </ol>
728  *
729  * <p>Note that since the Calendar class uses int to
730  * hold the value of year and month, this method may produce
731  * an unexpected result if this duration object holds
732  * a very large value in the years or months fields.</p>

```

```

732     *
733     * @param startInstant <code>Calendar</code> reference point.
734     *
735     * @return <code>Duration</code> of years and months of this <code>Duration</code> as days.
736     *
737     * @throws NullPointerException If the startInstant parameter is null.
738     */
739     public abstract Duration normalizeWith(final Calendar startInstant);
740
741     /**
742     * <p>Partial order relation comparison with this <code>Duration</code> instance.</p>
743     *
744     * <p>Comparison result must be in accordance with
745     * <a href="http://www.w3.org/TR/xmlschema-2/#duration-order">W3C XML Schema 1.0 Part 2,
746     *   Section 3.2.7.6.2,
747     * <i>Order relation on duration</i></a>.</p>
748     *
749     * <p>Return:</p>
750     * <ul>
751     *   <li>{@link DatatypeConstants#LESSER} if this <code>Duration</code> is shorter than <code>
752     *     duration</code> parameter</li>
753     *   <li>{@link DatatypeConstants#EQUAL} if this <code>Duration</code> is equal to <code>
754     *     duration</code> parameter</li>
755     *   <li>{@link DatatypeConstants#GREATER} if this <code>Duration</code> is longer than <code>
756     *     duration</code> parameter</li>
757     *   <li>{@link DatatypeConstants#INDETERMINATE} if a conclusive partial order relation cannot
758     *     be determined</li>
759     * </ul>
760     *
761     * @param duration to compare
762     *
763     * @return the relationship between <code>this</code> <code>Duration</code> and <code>duration</code>
764     *   parameter as
765     *   { {@link DatatypeConstants#LESSER}, {@link DatatypeConstants#EQUAL}, {@link
766     *     DatatypeConstants#GREATER}
767     *   or {@link DatatypeConstants#INDETERMINATE} }.
768     *
769     * @throws UnsupportedOperationException If the underlying implementation
770     *   cannot reasonably process the request, e.g. W3C XML Schema allows for
771     *   arbitrarily large/small/precise values, the request may be beyond the
772     *   implementations capability.
773     * @throws NullPointerException if <code>duration</code> is <code>null</code>.
774     *
775     * @see #isShorterThan(Duration)
776     * @see #isLongerThan(Duration)
777     */
778     public abstract int compare(final Duration duration);
779
780     /**
781     * <p>Checks if this duration object is strictly longer than
782     * another <code>Duration</code> object.</p>
783     *
784     * <p>Duration X is "longer" than Y if and only if X>Y

```

```

778 * as defined in the section 3.2.6.2 of the XML Schema 1.0
779 * specification.</p>
780 *
781 * <p>For example, "P1D" (one day) > "PT12H" (12 hours) and
782 * "P2Y" (two years) > "P23M" (23 months).</p>
783 *
784 * @param duration <code>Duration</code> to test this <code>Duration</code> against.
785 *
786 * @throws UnsupportedOperationException If the underlying implementation
787 * cannot reasonably process the request, e.g. W3C XML Schema allows for
788 * arbitrarily large/small/precise values, the request may be beyond the
789 * implementations capability.
790 * @throws NullPointerException If <code>duration</code> is null.
791 *
792 * @return
793 *     true if the duration represented by this object
794 *     is longer than the given duration. false otherwise.
795 *
796 * @see #isShorterThan(Duration)
797 * @see #compare(Duration duration)
798 */
799 public boolean isLongerThan(final Duration duration) {
800     return compare(duration) == DatatypeConstants.GREATER;
801 }
802
803 /**
804 * <p>Checks if this duration object is strictly shorter than
805 * another <code>Duration</code> object.</p>
806 *
807 * @param duration <code>Duration</code> to test this <code>Duration</code> against.
808 *
809 * @return <code>true</code> if <code>duration</code> parameter is shorter than this <code>
810 *     Duration</code>,
811 *     else <code>false</code>.
812 *
813 * @throws UnsupportedOperationException If the underlying implementation
814 * cannot reasonably process the request, e.g. W3C XML Schema allows for
815 * arbitrarily large/small/precise values, the request may be beyond the
816 * implementations capability.
817 * @throws NullPointerException if <code>duration</code> is null.
818 *
819 * @see #isLongerThan(Duration duration)
820 * @see #compare(Duration duration)
821 */
822 public boolean isShorterThan(final Duration duration) {
823     return compare(duration) == DatatypeConstants.LESSER;
824 }
825
826 /**
827 * <p>Checks if this duration object has the same duration
828 * as another <code>Duration</code> object.</p>
829 *
830 * <p>For example, "P1D" (1 day) is equal to "PT24H" (24 hours).</p>

```

```

830      *
831      * <p>Duration X is equal to Y if and only if time instant
832      * t+X and t+Y are the same for all the test time instants
833      * specified in the section 3.2.6.2 of the XML Schema 1.0
834      * specification.</p>
835      *
836      * <p>Note that there are cases where two <code>Duration</code>s are
837      * "incomparable" to each other, like one month and 30 days.
838      * For example,</p>
839      * <pre>
840      * !new Duration("P1M").isShorterThan(new Duration("P30D"))
841      * !new Duration("P1M").isLongerThan(new Duration("P30D"))
842      * !new Duration("P1M").equals(new Duration("P30D"))
843      * </pre>
844      *
845      * @param duration
846      *     The object to compare this <code>Duration</code> against.
847      *
848      * @return
849      *     <code>true</code> if this duration is the same length as
850      *     <code>duration</code>.
851      *     <code>false</code> if <code>duration</code> is <code>null</code>,
852      *     is not a
853      *     <code>Duration</code> object,
854      *     or its length is different from this duration.
855      *
856      * @throws UnsupportedOperationException If the underlying implementation
857      *     cannot reasonably process the request, e.g. W3C XML Schema allows for
858      *     arbitrarily large/small/precise values, the request may be beyond the
859      *     implementations capability.
860      *
861      * @see #compare(Duration duration)
862      */
863     public boolean equals(final Object duration) {
864
865         if (duration == null || !(duration instanceof Duration)) {
866             return false;
867         }
868
869         return compare((Duration) duration) == DatatypeConstants.EQUAL;
870     }
871
872     /**
873      * Returns a hash code consistent with the definition of the equals method.
874      *
875      * @see Object#hashCode()
876      */
877     public abstract int hashCode();
878
879     /**
880      * <p>Returns a <code>String</code> representation of this <code>Duration</code> <code>Object</code>.</p>
881      *

```

```

882 * <p>The result is formatted according to the XML Schema 1.0 spec and can be always parsed
      back later into the
883 * equivalent <code>Duration</code> <code>Object</code> by {@link DatatypeFactory#newDuration(
      String lexicalRepresentation)}.</p>
884 *
885 * <p>Formally, the following holds for any <code>Duration</code>
886 * <code>Object</code> x:</p>
887 * <pre>
888 * new Duration(x.toString()).equals(x)
889 * </pre>
890 *
891 * @return A non-<code>null</code> valid <code>String</code> representation of this <code>
      Duration</code>.
892 */
893 public String toString() {
894
895     StringBuffer buf = new StringBuffer();
896
897     if (getSign() < 0) {
898         buf.append('-');
899     }
900     buf.append('P');
901
902     BigInteger years = (BigInteger) getField(DatatypeConstants.YEARS);
903     if (years != null) {
904         buf.append(years + "Y");
905     }
906
907     BigInteger months = (BigInteger) getField(DatatypeConstants.MONTHS);
908     if (months != null) {
909         buf.append(months + "M");
910     }
911
912     BigInteger days = (BigInteger) getField(DatatypeConstants.DAYS);
913     if (days != null) {
914         buf.append(days + "D");
915     }
916
917     BigInteger hours = (BigInteger) getField(DatatypeConstants.HOURS);
918     BigInteger minutes = (BigInteger) getField(DatatypeConstants.MINUTES);
919     BigDecimal seconds = (BigDecimal) getField(DatatypeConstants.SECONDS);
920     if (hours != null || minutes != null || seconds != null) {
921         buf.append('T');
922         if (hours != null) {
923             buf.append(hours + "H");
924         }
925         if (minutes != null) {
926             buf.append(minutes + "M");
927         }
928         if (seconds != null) {
929             buf.append(toString(seconds) + "S");
930         }
931     }

```



```

932     return buf.toString();
933 }
934
935 /**
936  * <p>Turns {@link BigDecimal} to a string representation.</p>
937  *
938  * <p>Due to a behavior change in the {@link BigDecimal#toString()}
939  * method in JDK1.5, this had to be implemented here.</p>
940  *
941  * @param bd <code>BigDecimal</code> to format as a <code>String</code>
942  *
943  * @return <code>String</code> representation of <code>BigDecimal</code>
944  */
945 private String toString(BigDecimal bd) {
946     String intString = bd.unscaledValue().toString();
947     int scale = bd.scale();
948
949     if (scale == 0) {
950         return intString;
951     }
952
953     /* Insert decimal point */
954     StringBuffer buf;
955     int insertionPoint = intString.length() - scale;
956     if (insertionPoint == 0) { /* Point goes right before intVal */
957         return "0." + intString;
958     } else if (insertionPoint > 0) { /* Point goes inside intVal */
959         buf = new StringBuffer(intString);
960         buf.insert(insertionPoint, '.');
961     } else { /* We must insert zeros between point and intVal */
962         buf = new StringBuffer(3 - insertionPoint + intString.length());
963         buf.append("0.");
964         for (int i = 0; i < -insertionPoint; i++) {
965             buf.append('0');
966         }
967         buf.append(intString);
968     }
969     return buf.toString();
970 }
971
972 /**
973  * <p>Calls the {@link Calendar#getTimeInMillis} method.
974  * Prior to JDK1.4, this method was protected and therefore
975  * cannot be invoked directly.</p>
976  *
977  * <p>TODO: In future, this should be replaced by <code>cal.getTimeInMillis()</code>.</p>
978  *
979  * @param cal <code>Calendar</code> to get time in milliseconds.
980  *
981  * @return Milliseconds of <code>cal</code>.
982  */

```

```
985     private static long getCalendarTimeInMillis(final Calendar cal) {
986         return cal.getTime().getTime();
987     }
988 }
```

## 14.5 FactoryFinder.java

Secuencia 149: javax/xml/datatype/FactoryFinder.java

```
1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.datatype;
27
28 import java.io.File;
29 import java.security.AccessController;
30 import java.security.PrivilegedAction;
31 import java.util.Iterator;
32 import java.util.Properties;
33 import java.util.ServiceConfigurationError;
34 import java.util.ServiceLoader;
35
36 /**
37  * <p>Implements pluggable Datatypes.</p>
38  *
39  * <p>This class is duplicated for each JAXP subpackage so keep it in
40  * sync. It is package private for secure class loading.</p>
41  *
42  * @author Santiago.PericasGeertsen@sun.com
43  */
44 class FactoryFinder {
45     private static final String DEFAULT_PACKAGE = "com.sun.org.apache.xerces.internal";
```

```
46
47  /**
48   * Internal debug flag.
49   */
50  private static boolean debug = false;
51
52  /**
53   * Cache for properties in java.home/lib/jaxp.properties
54   */
55  private final static Properties cacheProps = new Properties();
56
57  /**
58   * Flag indicating if properties from java.home/lib/jaxp.properties
59   * have been cached.
60   */
61  private static volatile boolean firstTime = true;
62
63  /**
64   * Security support class use to check access control before
65   * getting certain system resources.
66   */
67  private final static SecuritySupport ss = new SecuritySupport();
68
69  // Define system property "jaxp.debug" to get output
70  static {
71      // Use try/catch block to support applets, which throws
72      // SecurityException out of this code.
73      try {
74          String val = ss.getSystemProperty("jaxp.debug");
75          // Allow simply setting the prop to turn on debug
76          debug = val != null && !"false".equals(val);
77      }
78      catch (SecurityException se) {
79          debug = false;
80      }
81  }
82
83  private static void dPrint(String msg) {
84      if (debug) {
85          System.err.println("JAXP: " + msg);
86      }
87  }
88
89  /**
90   * Attempt to load a class using the class loader supplied. If that fails
91   * and fall back is enabled, the current (i.e. bootstrap) class loader is
92   * tried.
93   *
94   * If the class loader supplied is <code>null</code>, first try using the
95   * context class loader followed by the current (i.e. bootstrap) class
96   * loader.
97   *
98   * Use bootstrap classLoader if cl = null and useBSClsLoader is true
```

```

99      */
100     static private Class<?> getProviderClass(String className, ClassLoader cl,
101         boolean doFallback, boolean useBSClsLoader) throws ClassNotFoundException
102     {
103         try {
104             if (cl == null) {
105                 if (useBSClsLoader) {
106                     return Class.forName(className, false, FactoryFinder.class.getClassLoader());
107                 } else {
108                     cl = ss.getContextClassLoader();
109                     if (cl == null) {
110                         throw new ClassNotFoundException();
111                     }
112                     else {
113                         return Class.forName(className, false, cl);
114                     }
115                 }
116             }
117             else {
118                 return Class.forName(className, false, cl);
119             }
120         }
121         catch (ClassNotFoundException e1) {
122             if (doFallback) {
123                 // Use current class loader - should always be bootstrap CL
124                 return Class.forName(className, false, FactoryFinder.class.getClassLoader());
125             }
126             else {
127                 throw e1;
128             }
129         }
130     }
131
132     /**
133      * Create an instance of a class. Delegates to method
134      * <code>getProviderClass()</code> in order to load the class.
135      *
136      * @param type Base class / Service interface of the factory to
137      *             instantiate.
138      *
139      * @param className Name of the concrete class corresponding to the
140      * service provider
141      *
142      * @param cl <code>ClassLoader</code> used to load the factory class. If <code>null</code>
143      * current <code>Thread</code>'s context classLoader is used to load the factory class.
144      *
145      * @param doFallback True if the current ClassLoader should be tried as
146      * a fallback if the class is not found using cl
147      */
148     static <T> T newInstance(Class<T> type, String className, ClassLoader cl, boolean doFallback)
149         throws DatatypeConfigurationException
150     {
151         return newInstance(type, className, cl, doFallback, false);

```

```

152     }
153
154     /**
155      * Create an instance of a class. Delegates to method
156      * <code>getProviderClass()</code> in order to load the class.
157      *
158      * @param type Base class / Service interface of the factory to
159      *             instantiate.
160      *
161      * @param className Name of the concrete class corresponding to the
162      *             service provider
163      *
164      * @param cl ClassLoader to use to load the class, null means to use
165      *             the bootstrap ClassLoader
166      *
167      * @param doFallback True if the current ClassLoader should be tried as
168      *             a fallback if the class is not found using cl
169      *
170      * @param useBSClsLoader True if cl=null actually meant bootstrap classLoader. This parameter
171      *             is needed since DocumentBuilderFactory/SAXParserFactory defined null as context classLoader.
172      */
173     static <T> T newInstance(Class<T> type, String className, ClassLoader cl,
174                             boolean doFallback, boolean useBSClsLoader)
175         throws DatatypeConfigurationException
176     {
177         assert type != null;
178
179         // make sure we have access to restricted packages
180         if (System.getSecurityManager() != null) {
181             if (className != null && className.startsWith(DEFAULT_PACKAGE)) {
182                 cl = null;
183                 useBSClsLoader = true;
184             }
185         }
186
187         try {
188             Class<?> providerClass = getProviderClass(className, cl, doFallback, useBSClsLoader);
189             if (!type.isAssignableFrom(providerClass)) {
190                 throw new ClassCastException(className + " cannot be cast to " + type.getName());
191             }
192             Object instance = providerClass.newInstance();
193             if (debug) { // Extra check to avoid computing cl strings
194                 dPrint("created new instance of " + providerClass +
195                     " using ClassLoader: " + cl);
196             }
197             return type.cast(instance);
198         }
199         catch (ClassNotFoundException x) {
200             throw new DatatypeConfigurationException(
201                 "Provider " + className + " not found", x);
202         }
203         catch (Exception x) {
204             throw new DatatypeConfigurationException(

```

```

205         "Provider " + className + " could not be instantiated: " + x,
206         x);
207     }
208 }
209
210 /**
211  * Finds the implementation Class object in the specified order. Main
212  * entry point.
213  * @return Class object of factory, never null
214  *
215  * @param type          Base class / Service interface of the
216  *                      factory to find.
217  * @param fallbackClassName Implementation class name, if nothing else
218  *                      is found. Use null to mean no fallback.
219  *
220  * Package private so this code can be shared.
221  */
222 static <T> T find(Class<T> type, String fallbackClassName)
223     throws DatatypeConfigurationException
224 {
225     final String factoryId = type.getName();
226     dPrint("find factoryId =" + factoryId);
227
228     // Use the system property first
229     try {
230         String systemProp = ss.getProperty(factoryId);
231         if (systemProp != null) {
232             dPrint("found system property, value=" + systemProp);
233             return newInstance(type, systemProp, null, true);
234         }
235     }
236     catch (SecurityException se) {
237         if (debug) se.printStackTrace();
238     }
239
240     // try to read from $java.home/lib/jaxp.properties
241     try {
242         if (firstTime) {
243             synchronized (cacheProps) {
244                 if (firstTime) {
245                     String configFile = ss.getProperty("java.home") + File.separator +
246                         "lib" + File.separator + "jaxp.properties";
247                     File f = new File(configFile);
248                     firstTime = false;
249                     if (ss.exists(f)) {
250                         dPrint("Read properties file "+f);
251                         cacheProps.load(ss.getInputStream(f));
252                     }
253                 }
254             }
255         }
256         final String factoryClassName = cacheProps.getProperty(factoryId);
257

```

```

258         if (factoryClassName != null) {
259             dPrint("found in $java.home/jaxp.properties, value=" + factoryClassName);
260             return newInstance(type, factoryClassName, null, true);
261         }
262     }
263     catch (Exception ex) {
264         if (debug) ex.printStackTrace();
265     }
266
267     // Try Jar Service Provider Mechanism
268     final T provider = findServiceProvider(type);
269     if (provider != null) {
270         return provider;
271     }
272     if (fallbackClassName == null) {
273         throw new DatatypeConfigurationException(
274             "Provider for " + factoryId + " cannot be found");
275     }
276
277     dPrint("loaded from fallback value: " + fallbackClassName);
278     return newInstance(type, fallbackClassName, null, true);
279 }
280
281 /*
282  * Try to find provider using the ServiceLoader API
283  *
284  * @param type Base class / Service interface of the factory to find.
285  *
286  * @return instance of provider class if found or null
287  */
288 private static <T> T findServiceProvider(final Class<T> type)
289     throws DatatypeConfigurationException
290 {
291     try {
292         return AccessController.doPrivileged(new PrivilegedAction<T>() {
293             public T run() {
294                 final ServiceLoader<T> serviceLoader = ServiceLoader.load(type);
295                 final Iterator<T> iterator = serviceLoader.iterator();
296                 if (iterator.hasNext()) {
297                     return iterator.next();
298                 } else {
299                     return null;
300                 }
301             }
302         });
303     } catch (ServiceConfigurationError e) {
304         final DatatypeConfigurationException error =
305             new DatatypeConfigurationException(
306                 "Provider for " + type + " cannot be found", e);
307         throw error;
308     }
309 }
310

```

311 }

## 14.6 SecuritySupport.java

Secuencia 150: javax/xml/datatype/SecuritySupport.java

```
1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.datatype;
27
28 import java.security.*;
29 import java.net.*;
30 import java.io.*;
31 import java.util.*;
32
33 /**
34 * This class is duplicated for each JAXP subpackage so keep it in sync.
35 * It is package private and therefore is not exposed as part of the JAXP
36 * API.
37 *
38 * Security related methods that only work on J2SE 1.2 and newer.
39 */
40 class SecuritySupport {
41
42     ClassLoader getContextClassLoader() {
43         return (ClassLoader)
44             AccessController.doPrivileged(new PrivilegedAction() {
45                 public Object run() {
46                     ClassLoader cl = null;
47                     try {
```



```

49         cl = Thread.currentThread().getContextClassLoader();
50     } catch (SecurityException ex) { }
51     return cl;
52 }
53 });
54 }
55
56 String getSystemProperty(final String propName) {
57     return (String)
58         AccessController.doPrivileged(new PrivilegedAction() {
59             public Object run() {
60                 return System.getProperty(propName);
61             }
62         });
63 }
64
65 FileInputStream getFileInputStream(final File file)
66     throws FileNotFoundException
67 {
68     try {
69         return (FileInputStream)
70             AccessController.doPrivileged(new PrivilegedExceptionAction() {
71                 public Object run() throws FileNotFoundException {
72                     return new FileInputStream(file);
73                 }
74             });
75     } catch (PrivilegedActionException e) {
76         throw (FileNotFoundException)e.getException();
77     }
78 }
79
80 InputStream getResourceAsStream(final ClassLoader cl,
81                                 final String name)
82 {
83     return (InputStream)
84         AccessController.doPrivileged(new PrivilegedAction() {
85             public Object run() {
86                 InputStream ris;
87                 if (cl == null) {
88                     ris = Object.class.getResourceAsStream(name);
89                 } else {
90                     ris = cl.getResourceAsStream(name);
91                 }
92                 return ris;
93             }
94         });
95 }
96
97 boolean doesFileExist(final File f) {
98     return ((Boolean)
99         AccessController.doPrivileged(new PrivilegedAction() {
100             public Object run() {
101                 return new Boolean(f.exists());

```

```

102     }
103     })).booleanValue();
104 }
105
106 }

```

## 14.7 XMLGregorianCalendar.java

Secuencia 151: javax/xml/datatype/XMLGregorianCalendar.java

```

1  /*
2  * Copyright (c) 2003, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.datatype;
27
28 import javax.xml.namespace.QName;
29 import java.math.BigDecimal;
30 import java.math.BigInteger;
31 import java.util.TimeZone;
32 import java.util.GregorianCalendar;
33
34 /**
35 * <p>Representation for W3C XML Schema 1.0 date/time datatypes.
36 * Specifically, these date/time datatypes are
37 * {@link DatatypeConstants#DATETIME},
38 * {@link DatatypeConstants#TIME},
39 * {@link DatatypeConstants#DATE},
40 * {@link DatatypeConstants#GYEARMONTH},
41 * {@link DatatypeConstants#GMONTHDAY},
42 * {@link DatatypeConstants#GYEAR},
43 * {@link DatatypeConstants#GMONTH}, and
44 * {@link DatatypeConstants#GDAY}

```

```

45 * defined in the XML Namespace
46 * <code>"http://www.w3.org/2001/XMLSchema"</code>.
47 * These datatypes are normatively defined in
48 * <a href="http://www.w3.org/TR/xmlschema-2/#dateTime">W3C XML Schema 1.0 Part 2, Section
    3.2.7-14</a>.</p>
49 *
50 * <p>The table below defines the mapping between XML Schema 1.0
51 * date/time datatype fields and this class' fields. It also summarizes
52 * the value constraints for the date and time fields defined in
53 * <a href="http://www.w3.org/TR/xmlschema-2/#isoformats">W3C XML Schema 1.0 Part 2, Appendix D,
54 * <i>ISO 8601 Date and Time Formats</i></a>.</p>
55 *
56 * <a name="datetimefieldmapping"/>
57 * <table border="2" rules="all" cellpadding="2">
58 *   <thead>
59 *     <tr>
60 *       <th align="center" colspan="3">
61 *         Date/Time Datatype Field Mapping Between XML Schema 1.0 and Java Representation
62 *       </th>
63 *     </tr>
64 *   </thead>
65 *   <tbody>
66 *     <tr>
67 *       <th>XML Schema 1.0<br/>
68 *         datatype<br/>
69 *         field</th>
70 *       <th>Related<br/>XMLGregorianCalendar<br/>Accessor(s)</th>
71 *       <th>Value Range</th>
72 *     </tr>
73 *     <tr>
74 *       <td><a name="datetimefield-year"/>year</td>
75 *       <td>{@link #getYear()} + {@link #getEon()} or<br/>
76 *         {@link #getEonAndYear}
77 *       </td>
78 *       <td><code>getYear()</code> is a value between  $-(10^9-1)$  to  $(10^9)-1$ 
79 *         or {@link DatatypeConstants#FIELD_UNDEFINED}.<br/>
80 *         {@link #getEon()} is high order year value in billion of years.<br/>
81 *         <code>getEon()</code> has values greater than or equal to  $(10^9)$  or less than or
82 *         equal to  $-(10^9)$ .
83 *         A value of null indicates field is undefined.<br/>
84 *         Given that <a href="http://www.w3.org/2001/05/xmlschema-errata#e2-63">XML Schema 1.0
85 *         errata</a> states that the year zero
86 *         will be a valid lexical value in a future version of XML Schema,
87 *         this class allows the year field to be set to zero. Otherwise,
88 *         the year field value is handled exactly as described
89 *         in the errata and [ISO-8601-1988]. Note that W3C XML Schema 1.0
90 *         validation does not allow for the year field to have a value of zero.
91 *       </td>
92 *     </tr>
93 *     <tr>
94 *       <td><a name="datetimefield-month"/>month</td>
95 *       <td>{@link #getMonth()} </td>
96 *       <td>1 to 12 or {@link DatatypeConstants#FIELD_UNDEFINED} </td>

```

```

95 *      </tr>
96 *      <tr>
97 *          <td><a name="datetimefield-day"/>day</td>
98 *          <td>{@link #getDay()} </td>
99 *          <td> Independent of month, max range is 1 to 31 or {@link DatatypeConstants#
FIELD_UNDEFINED}.<br/>
100 *              The normative value constraint stated relative to month
101 *              field's value is in <a href="http://www.w3.org/TR/xmlschema-2/#isoformats">W3C XML
Schema 1.0 Part 2, Appendix D</a>.
102 *          </td>
103 *      </tr>
104 *      <tr>
105 *          <td><a name="datetimefield-hour"/>hour</td>
106 *          <td>{@link #getHour()}</td>
107 *          <td>
108 *              0 to 23 or {@link DatatypeConstants#FIELD_UNDEFINED}.
109 *              An hour value of 24 is allowed to be set in the lexical space provided the minute and
second
110 *              field values are zero. However, an hour value of 24 is not allowed in value space and
will be
111 *              transformed to represent the value of the first instance of the following day as per
112 *              <a href="http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes">
113 *              XML Schema Part 2: Datatypes Second Edition, 3.2 Primitive datatypes</a>.
114 *          </td>
115 *      </tr>
116 *      <tr>
117 *          <td><a name="datetimefield-minute"/>minute</td>
118 *          <td>{@link #getMinute()} </td>
119 *          <td> 0 to 59 or {@link DatatypeConstants#FIELD_UNDEFINED} </td>
120 *      </tr>
121 *      <tr>
122 *          <td><a name="datetimefield-second"/>second</td>
123 *          <td>
124 *              {@link #getSecond()} + {@link #getMillisecond()}/1000 or<br/>
125 *              {@link #getSecond()} + {@link #getFractionalSecond()}
126 *          </td>
127 *          <td>
128 *              {@link #getSecond()} from 0 to 60 or {@link DatatypeConstants#FIELD_UNDEFINED}.<br/>
129 *              <i>(Note: 60 only allowable for leap second.)</i><br/>
130 *              {@link #getFractionalSecond()} allows for infinite precision over the range from 0.0 to
1.0 when
131 *              the {@link #getSecond()} is defined.<br/>
132 *              <code>FractionalSecond</code> is optional and has a value of <code>null</code> when it
is undefined.<br />
133 *              {@link #getMillisecond()} is the convenience
134 *              millisecond precision of value of {@link #getFractionalSecond()}.
135 *          </td>
136 *      </tr>
137 *      <tr>
138 *          <td><a name="datetimefield-timezone"/>timezone</td>
139 *          <td>{@link #getTimezone()} </td>
140 *          <td> Number of minutes or {@link DatatypeConstants#FIELD_UNDEFINED}.
141 *          Value range from -14 hours (-14 * 60 minutes) to 14 hours (14 * 60 minutes).

```

```

142 *      </td>
143 *      </tr>
144 *      </tbody>
145 *      </table>
146 *
147 * <p>All maximum value space constraints listed for the fields in the table
148 * above are checked by factory methods, {@link DatatypeFactory},
149 * setter methods and parse methods of
150 * this class. <code>IllegalArgumentException</code> is thrown when a
151 * parameter's value is outside the value constraint for the field or
152 * if the composite
153 * values constitute an invalid XMLGregorianCalendar instance (for example, if
154 * the 31st of June is specified).
155 * </p>
156 *
157 * <p>The following operations are defined for this class:
158 * <ul>
159 *   <li>accessors/mutators for independent date/time fields</li>
160 *   <li>conversion between this class and W3C XML Schema 1.0 lexical representation,
161 *     {@link #toString()}, {@link DatatypeFactory#newXMLGregorianCalendar(String
162 *       lexicalRepresentation)}</li>
163 *   <li>conversion between this class and {@link GregorianCalendar},
164 *     {@link #toGregorianCalendar(java.util.TimeZone timezone, java.util.Locale aLocale,
165 *       XMLGregorianCalendar defaults)},
166 *     {@link DatatypeFactory}</li>
167 *   <li>partial order relation comparator method, {@link #compare(XMLGregorianCalendar
168 *       xmlGregorianCalendar)}</li>
169 *   <li>{@link #equals(Object)} defined relative to {@link #compare(XMLGregorianCalendar
170 *       xmlGregorianCalendar)}.</li>
171 *   <li>addition operation with {@link Duration}
172 *     instance as defined in <a href="http://www.w3.org/TR/xmlschema-2/#adding-durations-to-
173 *       dateTimes">
174 *       W3C XML Schema 1.0 Part 2, Appendix E, <i>Adding durations to dateTimes</i></a>.
175 *   </li>
176 * </ul>
177 * </p>
178 *
179 * @author <a href="mailto:Joseph.Fialli@Sun.com">Joseph Fialli</a>
180 * @author <a href="mailto:Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
181 * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
182 * @author <a href="mailto:Sunitha.Reddy@Sun.com">Sunitha Reddy</a>
183 * @see Duration
184 * @see DatatypeFactory
185 * @since 1.5
186 */
187
188 public abstract class XMLGregorianCalendar
189     implements Cloneable {
190
191     /**
192      * Default no-arg constructor.
193      *
194      * <p>Note: Always use the {@link DatatypeFactory} to

```

```

190     * construct an instance of <code>XMLGregorianCalendar</code>.
191     * The constructor on this class cannot be guaranteed to
192     * produce an object with a consistent state and may be
193     * removed in the future.</p>
194     */
195     public XMLGregorianCalendar() {
196     }
197
198     /**
199     * <p>Unset all fields to undefined.</p>
200     *
201     * <p>Set all int fields to {@link DatatypeConstants#FIELD_UNDEFINED} and reference fields
202     * to null.</p>
203     */
204     public abstract void clear();
205
206     /**
207     * <p>Reset this <code>XMLGregorianCalendar</code> to its original values.</p>
208     *
209     * <p><code>XMLGregorianCalendar</code> is reset to the same values as when it was created
210     * with
211     * { {@link DatatypeFactory#newXMLGregorianCalendar()},
212     *   {@link DatatypeFactory#newXMLGregorianCalendar(String lexicalRepresentation)},
213     *   {@link DatatypeFactory#newXMLGregorianCalendar(
214     *     BigInteger year,
215     *     int month,
216     *     int day,
217     *     int hour,
218     *     int minute,
219     *     int second,
220     *     BigDecimal fractionalSecond,
221     *     int timezone)},
222     *   {@link DatatypeFactory#newXMLGregorianCalendar(
223     *     int year,
224     *     int month,
225     *     int day,
226     *     int hour,
227     *     int minute,
228     *     int second,
229     *     int millisecond,
230     *     int timezone)},
231     *   {@link DatatypeFactory#newXMLGregorianCalendar(GregorianCalendar cal)},
232     *   {@link DatatypeFactory#newXMLGregorianCalendarDate(
233     *     int year,
234     *     int month,
235     *     int day,
236     *     int timezone)},
237     *   {@link DatatypeFactory#newXMLGregorianCalendarTime(
238     *     int hours,
239     *     int minutes,
240     *     int seconds,
241     *     int timezone)},
242     *   {@link DatatypeFactory#newXMLGregorianCalendarTime(

```

```

242     *   int hours,
243     *   int minutes,
244     *   int seconds,
245     *   BigDecimal fractionalSecond,
246     *   int timezone)} or
247     *   {@link DatatypeFactory#newXMLGregorianCalendarTime(
248     *   int hours,
249     *   int minutes,
250     *   int seconds,
251     *   int milliseconds,
252     *   int timezone)}.
253     * </p>
254     *
255     * <p><code>reset()</code> is designed to allow the reuse of existing <code>
XMLGregorianCalendar</code>s
256     * thus saving resources associated with the creation of new <code>XMLGregorianCalendar</
code>s.</p>
257     */
258     public abstract void reset();
259
260     /**
261     * <p>Set low and high order component of XSD <code>dateTime</code> year field.</p>
262     *
263     * <p>Unset this field by invoking the setter with a parameter value of <code>null</code>.</p>
264     *
265     * @param year value constraints summarized in <a href="#datetimefield-year">year field of date
/time field mapping table</a>.
266     *
267     * @throws IllegalArgumentException if <code>year</code> parameter is
268     * outside value constraints for the field as specified in
269     * <a href="#datetimefieldmapping">date/time field mapping table</a>.
270     */
271     public abstract void setYear(BigInteger year);
272
273     /**
274     * <p>Set year of XSD <code>dateTime</code> year field.</p>
275     *
276     * <p>Unset this field by invoking the setter with a parameter value of
277     * {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
278     *
279     * <p>Note: if the absolute value of the <code>year</code> parameter
280     * is less than 109, the eon component of the XSD year field is set to
281     * <code>null</code> by this method.</p>
282     *
283     * @param year value constraints are summarized in <a href="#datetimefield-year">year field of
date/time field mapping table</a>.
284     * If year is {@link DatatypeConstants#FIELD_UNDEFINED}, then eon is set to <code>null</code>
>.
285     */
286     public abstract void setYear(int year);
287
288     /**
289     * <p>Set month.</p>

```

```

290 *
291 * <p>Unset this field by invoking the setter with a parameter value of {@link
    DatatypeConstants#FIELD_UNDEFINED}.</p>
292 *
293 * @param month value constraints summarized in <a href="#datetimefield-month">month field of
    date/time field mapping table</a>.
294 *
295 * @throws IllegalArgumentException if <code>month</code> parameter is
296 * outside value constraints for the field as specified in
297 * <a href="#datetimefieldmapping">date/time field mapping table</a>.
298 */
299 public abstract void setMonth(int month);
300
301 /**
302 * <p>Set days in month.</p>
303 *
304 * <p>Unset this field by invoking the setter with a parameter value of {@link
    DatatypeConstants#FIELD_UNDEFINED}.</p>
305 *
306 * @param day value constraints summarized in <a href="#datetimefield-day">day field of date/
    time field mapping table</a>.
307 *
308 * @throws IllegalArgumentException if <code>day</code> parameter is
309 * outside value constraints for the field as specified in
310 * <a href="#datetimefieldmapping">date/time field mapping table</a>.
311 */
312 public abstract void setDay(int day);
313
314 /**
315 * <p>Set the number of minutes in the timezone offset.</p>
316 *
317 * <p>Unset this field by invoking the setter with a parameter value of {@link
    DatatypeConstants#FIELD_UNDEFINED}.</p>
318 *
319 * @param offset value constraints summarized in <a href="#datetimefield-timezone">
    timezone field of date/time field mapping table</a>.
320 *
321 *
322 * @throws IllegalArgumentException if <code>offset</code> parameter is
323 * outside value constraints for the field as specified in
324 * <a href="#datetimefieldmapping">date/time field mapping table</a>.
325 */
326 public abstract void setTimezone(int offset);
327
328 /**
329 * <p>Set time as one unit.</p>
330 *
331 * @param hour value constraints are summarized in
332 * <a href="#datetimefield-hour">hour field of date/time field mapping table</a>.
333 * @param minute value constraints are summarized in
334 * <a href="#datetimefield-minute">minute field of date/time field mapping table</a>.
335 * @param second value constraints are summarized in
336 * <a href="#datetimefield-second">second field of date/time field mapping table</a>.
337 *

```



```

338     * @see #setTime(int, int, int, BigDecimal)
339     *
340     * @throws IllegalArgumentException if any parameter is
341     * outside value constraints for the field as specified in
342     * <a href="#datetimefieldmapping">date/time field mapping table</a>.
343     */
344     public void setTime(int hour, int minute, int second) {
345
346         setTime(
347             hour,
348             minute,
349             second,
350             null // fractional
351         );
352     }
353
354     /**
355     * <p>Set hours.</p>
356     *
357     * <p>Unset this field by invoking the setter with a parameter value of {@link
358     * DatatypeConstants#FIELD_UNDEFINED}.</p>
359     *
360     * @param hour value constraints summarized in <a href="#datetimefield-hour">hour field of
361     * date/time field mapping table</a>.
362     *
363     * @throws IllegalArgumentException if <code>hour</code> parameter is outside value
364     * constraints for the field as specified in
365     * <a href="#datetimefieldmapping">date/time field mapping table</a>.
366     */
367     public abstract void setHour(int hour);
368
369     /**
370     * <p>Set minutes.</p>
371     *
372     * <p>Unset this field by invoking the setter with a parameter value of {@link
373     * DatatypeConstants#FIELD_UNDEFINED}.</p>
374     *
375     * @param minute value constraints summarized in <a href="#datetimefield-minute">minute
376     * field of date/time field mapping table</a>.
377     *
378     * @throws IllegalArgumentException if <code>minute</code> parameter is outside value
379     * constraints for the field as specified in
380     * <a href="#datetimefieldmapping">date/time field mapping table</a>.
381     */
382     public abstract void setMinute(int minute);
383
384     /**
385     * <p>Set seconds.</p>
386     *
387     * <p>Unset this field by invoking the setter with a parameter value of {@link
388     * DatatypeConstants#FIELD_UNDEFINED}.</p>
389     *
390     * @param second value constraints summarized in <a href="#datetimefield-second">second

```

```

        field of date/time field mapping table</a>.
384     *
385     * @throws IllegalArgumentException if <code>second</code> parameter is outside value
        constraints for the field as specified in
386     * <a href="#datetimefieldmapping">date/time field mapping table</a>.
387     */
388     public abstract void setSecond(int second);
389
390     /**
391     * <p>Set milliseconds.</p>
392     *
393     * <p>Unset this field by invoking the setter with a parameter value of {@link
        DatatypeConstants#FIELD_UNDEFINED}</p>
394     *
395     * @param millisecond value constraints summarized in
396     * <a href="#datetimefield-second">second field of date/time field mapping table</a>.
397     *
398     * @throws IllegalArgumentException if <code>millisecond</code> parameter is outside value
        constraints for the field as specified
399     * in <a href="#datetimefieldmapping">date/time field mapping table</a>.
400     */
401     public abstract void setMillisecond(int millisecond);
402
403     /**
404     * <p>Set fractional seconds.</p>
405     *
406     * <p>Unset this field by invoking the setter with a parameter value of <code>null</code>
        >.</p>
407     *
408     * @param fractional value constraints summarized in
409     * <a href="#datetimefield-second">second field of date/time field mapping table</a>.
410     *
411     * @throws IllegalArgumentException if <code>fractional</code> parameter is outside value
        constraints for the field as specified
412     * in <a href="#datetimefieldmapping">date/time field mapping table</a>.
413     */
414     public abstract void setFractionalSecond(BigDecimal fractional);
415
416
417     /**
418     * <p>Set time as one unit, including the optional infinite precision
419     * fractional seconds.</p>
420     *
421     * @param hour value constraints are summarized in
422     * <a href="#datetimefield-hour">hour field of date/time field mapping table</a>.
423     * @param minute value constraints are summarized in
424     * <a href="#datetimefield-minute">minute field of date/time field mapping table</a>.
425     * @param second value constraints are summarized in
426     * <a href="#datetimefield-second">second field of date/time field mapping table</a>.
427     * @param fractional value of <code>null</code> indicates this optional
428     * field is not set.
429     *
430     * @throws IllegalArgumentException if any parameter is

```

```

431     * outside value constraints for the field as specified in
432     * <a href="#datetimefieldmapping">date/time field mapping table</a>.
433     */
434     public void setTime(
435         int hour,
436         int minute,
437         int second,
438         BigDecimal fractional) {
439
440         setHour(hour);
441         setMinute(minute);
442         setSecond(second);
443         setFractionalSecond(fractional);
444     }
445
446
447     /**
448     * <p>Set time as one unit, including optional milliseconds.</p>
449     *
450     * @param hour value constraints are summarized in
451     * <a href="#datetimefield-hour">hour field of date/time field mapping table</a>.
452     * @param minute value constraints are summarized in
453     * <a href="#datetimefield-minute">minute field of date/time field mapping table</a>.
454     * @param second value constraints are summarized in
455     * <a href="#datetimefield-second">second field of date/time field mapping table</a>.
456     * @param millisecond value of {@link DatatypeConstants#FIELD_UNDEFINED} indicates this
457     * optional field is not set.
458     *
459     * @throws IllegalArgumentException if any parameter is
460     * outside value constraints for the field as specified in
461     * <a href="#datetimefieldmapping">date/time field mapping table</a>.
462     */
463     public void setTime(int hour, int minute, int second, int millisecond) {
464
465         setHour(hour);
466         setMinute(minute);
467         setSecond(second);
468         setMillisecond(millisecond);
469     }
470
471     /**
472     * <p>Return high order component for XML Schema 1.0 dateTime datatype field for
473     * <code>year</code>.
474     * <code>null</code> if this optional part of the year field is not defined.</p>
475     *
476     * <p>Value constraints for this value are summarized in
477     * <a href="#datetimefield-year">year field of date/time field mapping table</a>.</p>
478     * @return eon of this <code>XMLGregorianCalendar</code>. The value
479     * returned is an integer multiple of 109.
480     *
481     * @see #getYear()
482     * @see #getEonAndYear()
483     */

```

```

484     public abstract BigInteger getEon();
485
486     /**
487      * <p>Return low order component for XML Schema 1.0 dateTime datatype field for
488      * <code>year</code> or {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
489      *
490      * <p>Value constraints for this value are summarized in
491      * <a href="#datetimefield-year">year field of date/time field mapping table</a>.</p>
492      *
493      * @return year of this <code>XMLGregorianCalendar</code>.
494      *
495      * @see #getEon()
496      * @see #getEonAndYear()
497      */
498     public abstract int getYear();
499
500     /**
501      * <p>Return XML Schema 1.0 dateTime datatype field for
502      * <code>year</code>.</p>
503      *
504      * <p>Value constraints for this value are summarized in
505      * <a href="#datetimefield-year">year field of date/time field mapping table</a>.</p>
506      *
507      * @return sum of <code>eon</code> and <code>BigInteger.valueOf(year)</code>
508      * when both fields are defined. When only <code>year</code> is defined,
509      * return it. When both <code>eon</code> and <code>year</code> are not
510      * defined, return <code>null</code>.
511      *
512      * @see #getEon()
513      * @see #getYear()
514      */
515     public abstract BigInteger getEonAndYear();
516
517     /**
518      * <p>Return number of month or {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
519      *
520      * <p>Value constraints for this value are summarized in
521      * <a href="#datetimefield-month">month field of date/time field mapping table</a>.</p>
522      *
523      * @return year of this <code>XMLGregorianCalendar</code>.
524      *
525      */
526     public abstract int getMonth();
527
528     /**
529      * Return day in month or {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
530      *
531      * <p>Value constraints for this value are summarized in
532      * <a href="#datetimefield-day">day field of date/time field mapping table</a>.</p>
533      *
534      * @see #setDay(int)
535      */
536     public abstract int getDay();

```

```

537
538 /**
539  * Return timezone offset in minutes or
540  * {@link DatatypeConstants#FIELD_UNDEFINED} if this optional field is not defined.
541  *
542  * <p>Value constraints for this value are summarized in
543  * <a href="#datetimefield-timezone">timezone field of date/time field mapping table</a>.</p>
544  *
545  * @see #setTimezone(int)
546  */
547 public abstract int getTimezone();
548
549 /**
550  * Return hours or {@link DatatypeConstants#FIELD_UNDEFINED}.
551  * Returns {@link DatatypeConstants#FIELD_UNDEFINED} if this field is not defined.
552  *
553  * <p>Value constraints for this value are summarized in
554  * <a href="#datetimefield-hour">hour field of date/time field mapping table</a>.</p>
555  * @see #setTime(int, int, int)
556  */
557 public abstract int getHour();
558
559 /**
560  * Return minutes or {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
561  * Returns {@link DatatypeConstants#FIELD_UNDEFINED} if this field is not defined.
562  *
563  * <p>Value constraints for this value are summarized in
564  * <a href="#datetimefield-minute">minute field of date/time field mapping table</a>.</p>
565  * @see #setTime(int, int, int)
566  */
567 public abstract int getMinute();
568
569 /**
570  * <p>Return seconds or {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
571  *
572  * <p>Returns {@link DatatypeConstants#FIELD_UNDEFINED} if this field is not defined.
573  * When this field is not defined, the optional xs:dateTime
574  * fractional seconds field, represented by
575  * {@link #getFractionalSecond()} and {@link #getMillisecond()},
576  * must not be defined.</p>
577  *
578  * <p>Value constraints for this value are summarized in
579  * <a href="#datetimefield-second">second field of date/time field mapping table</a>.</p>
580  *
581  * @return Second of this <code>XMLGregorianCalendar</code>.
582  *
583  * @see #getFractionalSecond()
584  * @see #getMillisecond()
585  * @see #setTime(int, int, int)
586  */
587 public abstract int getSecond();
588

```

```

589  /**
590   * <p>Return millisecond precision of {@link #getFractionalSecond()}.</p>
591   *
592   * <p>This method represents a convenience accessor to infinite
593   * precision fractional second value returned by
594   * {@link #getFractionalSecond()}. The returned value is the rounded
595   * down to milliseconds value of
596   * {@link #getFractionalSecond()}. When {@link #getFractionalSecond()}
597   * returns <code>null</code>, this method must return
598   * {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
599   *
600   * <p>Value constraints for this value are summarized in
601   * <a href="#datetimefield-second">second field of date/time field mapping table</a>.</p>
602   *
603   * @return Millisecond of this <code>XMLGregorianCalendar</code>.
604   *
605   * @see #getFractionalSecond()
606   * @see #setTime(int, int, int)
607   */
608  public int getMillisecond() {
609
610      BigDecimal fractionalSeconds = getFractionalSecond();
611
612      // is field undefined?
613      if (fractionalSeconds == null) {
614          return DatatypeConstants.FIELD_UNDEFINED;
615      }
616
617      return getFractionalSecond().movePointRight(3).intValue();
618  }
619
620  /**
621   * <p>Return fractional seconds.</p>
622   *
623   * <p><code>null</code> is returned when this optional field is not defined.</p>
624   *
625   * <p>Value constraints are detailed in
626   * <a href="#datetimefield-second">second field of date/time field mapping table</a>.</p>
627   *
628   * <p>This optional field can only have a defined value when the
629   * xs:dateTime second field, represented by {@link #getSecond()},
630   * does not return {@link DatatypeConstants#FIELD_UNDEFINED}.</p>
631   *
632   * @return fractional seconds of this <code>XMLGregorianCalendar</code>.
633   *
634   * @see #getSecond()
635   * @see #setTime(int, int, int, BigDecimal)
636   */
637  public abstract BigDecimal getFractionalSecond();
638
639  // comparisons
640  /**
641   * <p>Compare two instances of W3C XML Schema 1.0 date/time datatypes

```

```

642 * according to partial order relation defined in
643 * <a href="http://www.w3.org/TR/xmlschema-2/#dateTime-order">W3C XML Schema 1.0 Part 2,
    Section 3.2.7.3,
644 * <i>Order relation on dateTime</i></a>.</p>
645 *
646 * <p><code>xsd:dateTime</code> datatype field mapping to accessors of
647 * this class are defined in
648 * <a href="#datetimefieldmapping">date/time field mapping table</a>.</p>
649 *
650 * @param xmlGregorianCalendar Instance of <code>XMLGregorianCalendar</code> to compare
651 *
652 * @return The relationship between <code>this</code> <code>XMLGregorianCalendar</code> and
653 * the specified <code>xmlGregorianCalendar</code> as
654 * {@link DatatypeConstants#LESSER},
655 * {@link DatatypeConstants#EQUAL},
656 * {@link DatatypeConstants#GREATER} or
657 * {@link DatatypeConstants#INDETERMINATE}.
658 *
659 * @throws NullPointerException if <code>xmlGregorianCalendar</code> is null.
660 */
661 public abstract int compare(XMLGregorianCalendar xmlGregorianCalendar);
662
663 /**
664 * <p>Normalize this instance to UTC.</p>
665 *
666 * <p>2000-03-04T23:00:00+03:00 normalizes to 2000-03-04T20:00:00Z</p>
667 * <p>Implements W3C XML Schema Part 2, Section 3.2.7.3 (A).</p>
668 *
669 * @return <code>this</code> <code>XMLGregorianCalendar</code> normalized to UTC.
670 */
671 public abstract XMLGregorianCalendar normalize();
672
673 /**
674 * <p>Compares this calendar to the specified object. The result is
675 * <code>true</code> if and only if the argument is not null and is an
676 * <code>XMLGregorianCalendar</code> object that represents the same
677 * instant in time as this object.</p>
678 *
679 * @param obj to compare.
680 *
681 * @return <code>true</code> when <code>obj</code> is an instance of
682 * <code>XMLGregorianCalendar</code> and
683 * {@link #compare(XMLGregorianCalendar obj)}
684 * returns {@link DatatypeConstants#EQUAL},
685 * otherwise <code>>false</code>.
686 */
687 public boolean equals(Object obj) {
688
689     if (obj == null || !(obj instanceof XMLGregorianCalendar)) {
690         return false;
691     }
692     return compare((XMLGregorianCalendar) obj) == DatatypeConstants.EQUAL;
693 }

```

```

694
695 /**
696  * <p>Returns a hash code consistent with the definition of the equals method.</p>
697  *
698  * @return hash code of this object.
699  */
700 public int hashCode() {
701
702     // Following two dates compare to EQUALS since in different timezones.
703     // 2000-01-15T12:00:00-05:00 == 2000-01-15T13:00:00-04:00
704     //
705     // Must ensure both instances generate same hashCode by normalizing
706     // this to UTC timezone.
707     int timezone = getTimezone();
708     if (timezone == DatatypeConstants.FIELD_UNDEFINED) {
709         timezone = 0;
710     }
711     XMLGregorianCalendar gc = this;
712     if (timezone != 0) {
713         gc = this.normalize();
714     }
715     return gc.getYear()
716         + gc.getMonth()
717         + gc.getDay()
718         + gc.getHour()
719         + gc.getMinute()
720         + gc.getSecond();
721 }
722
723 /**
724  * <p>Return the lexical representation of <code>this</code> instance.
725  * The format is specified in
726  * <a href="http://www.w3.org/TR/xmlschema-2/#dateTime-order">XML Schema 1.0 Part 2, Section
727  * 3.2. [7-14].1,
728  * <i>Lexical Representation</i>.</a></p>
729  *
730  * <p>Specific target lexical representation format is determined by
731  * {<link #getXMLSchemaType()>}.</p>
732  *
733  * @return XML, as <code>String</code>, representation of this <code>XMLGregorianCalendar</code>
734  *
735  * @throws IllegalStateException if the combination of set fields
736  *     does not match one of the eight defined XML Schema builtin date/time datatypes.
737  */
738 public abstract String toXMLFormat();
739
740 /**
741  * <p>Return the name of the XML Schema date/time type that this instance
742  * maps to. Type is computed based on fields that are set.</p>
743  *
744  * <table border="2" rules="all" cellpadding="2">
745  *     <thead>

```



```

745 *      <tr>
746 *          <th align="center" colspan="7">
747 *              Required fields for XML Schema 1.0 Date/Time Datatypes.<br/>
748 *              <i>(timezone is optional for all date/time datatypes)</i>
749 *          </th>
750 *      </tr>
751 *  </thead>
752 *  <tbody>
753 *      <tr>
754 *          <td>Datatype</td>
755 *          <td>year</td>
756 *          <td>month</td>
757 *          <td>day</td>
758 *          <td>hour</td>
759 *          <td>minute</td>
760 *          <td>second</td>
761 *      </tr>
762 *      <tr>
763 *          <td>{@link DatatypeConstants#DATETIME}</td>
764 *          <td>X</td>
765 *          <td>X</td>
766 *          <td>X</td>
767 *          <td>X</td>
768 *          <td>X</td>
769 *          <td>X</td>
770 *      </tr>
771 *      <tr>
772 *          <td>{@link DatatypeConstants#DATE}</td>
773 *          <td>X</td>
774 *          <td>X</td>
775 *          <td>X</td>
776 *          <td></td>
777 *          <td></td>
778 *          <td></td>
779 *      </tr>
780 *      <tr>
781 *          <td>{@link DatatypeConstants#TIME}</td>
782 *          <td></td>
783 *          <td></td>
784 *          <td></td>
785 *          <td>X</td>
786 *          <td>X</td>
787 *          <td>X</td>
788 *      </tr>
789 *      <tr>
790 *          <td>{@link DatatypeConstants#GYEARMONTH}</td>
791 *          <td>X</td>
792 *          <td>X</td>
793 *          <td></td>
794 *          <td></td>
795 *          <td></td>
796 *          <td></td>
797 *      </tr>

```

```

798 *      <tr>
799 *          <td>{@link DatatypeConstants#GMONTHDAY}</td>
800 *          <td></td>
801 *          <td>X</td>
802 *          <td>X</td>
803 *          <td></td>
804 *          <td></td>
805 *          <td></td>
806 *      </tr>
807 *      <tr>
808 *          <td>{@link DatatypeConstants#GYEAR}</td>
809 *          <td>X</td>
810 *          <td></td>
811 *          <td></td>
812 *          <td></td>
813 *          <td></td>
814 *          <td></td>
815 *      </tr>
816 *      <tr>
817 *          <td>{@link DatatypeConstants#GMONTH}</td>
818 *          <td></td>
819 *          <td>X</td>
820 *          <td></td>
821 *          <td></td>
822 *          <td></td>
823 *          <td></td>
824 *      </tr>
825 *      <tr>
826 *          <td>{@link DatatypeConstants#GDAY}</td>
827 *          <td></td>
828 *          <td></td>
829 *          <td>X</td>
830 *          <td></td>
831 *          <td></td>
832 *          <td></td>
833 *      </tr>
834 *  </tbody>
835 * </table>
836 *
837 * @throws java.lang.IllegalStateException if the combination of set fields
838 *      does not match one of the eight defined XML Schema builtin
839 *      date/time datatypes.
840 * @return One of the following class constants:
841 *      {@link DatatypeConstants#DATETIME},
842 *      {@link DatatypeConstants#TIME},
843 *      {@link DatatypeConstants#DATE},
844 *      {@link DatatypeConstants#GYEARMONTH},
845 *      {@link DatatypeConstants#GMONTHDAY},
846 *      {@link DatatypeConstants#GYEAR},
847 *      {@link DatatypeConstants#GMONTH} or
848 *      {@link DatatypeConstants#GDAY}.
849 */
850 public abstract QName getXMLSchemaType();

```

```

851
852     /**
853      * <p>Returns a <code>String</code> representation of this <code>XMLGregorianCalendar</code>
854      * > <code>Object</code>.</p>
855      *
856      * <p>The result is a lexical representation generated by {@link #toXMLFormat()}.</p>
857      *
858      * @return A non-<code>null</code> valid <code>String</code> representation of this <code>
859      * XMLGregorianCalendar</code>.
860      *
861      * @throws IllegalStateException if the combination of set fields
862      * does not match one of the eight defined XML Schema builtin date/time datatypes.
863      *
864      * @see #toXMLFormat()
865      */
866     public String toString() {
867         return toXMLFormat();
868     }
869
870     /**
871      * Validate instance by <code>getXMLSchemaType()</code> constraints.
872      * @return true if data values are valid.
873      */
874     public abstract boolean isValid();
875
876     /**
877      * <p>Add <code>duration</code> to this instance.</p>
878      *
879      * <p>The computation is specified in
880      * <a href="http://www.w3.org/TR/xmlschema-2/#adding-durations-to-dateTimes">XML Schema 1.0
881      * Part 2, Appendix E,
882      * <i>Adding durations to dateTimes</i></a>.
883      * <a href="#datetimefieldmapping">date/time field mapping table</a>
884      * defines the mapping from XML Schema 1.0 <code>dateTime</code> fields
885      * to this class' representation of those fields.</p>
886      *
887      * @param duration Duration to add to this <code>XMLGregorianCalendar</code>.
888      *
889      * @throws NullPointerException when <code>duration</code> parameter is <code>null</code>.
890      */
891     public abstract void add(Duration duration);
892
893     /**
894      * <p>Convert this <code>XMLGregorianCalendar</code> to a {@link GregorianCalendar}.</p>
895      *
896      * <p>When <code>this</code> instance has an undefined field, this
897      * conversion relies on the <code>java.util.GregorianCalendar</code> default
898      * for its corresponding field. A notable difference between
899      * XML Schema 1.0 date/time datatypes and <code>java.util.GregorianCalendar</code>
900      * is that Timezone value is optional for date/time datatypes and it is
901      * a required field for <code>java.util.GregorianCalendar</code>. See javadoc
902      * for <code>java.util.TimeZone.getDefault()</code> on how the default

```

```

901 * is determined. To explicitly specify the <code>TimeZone</code>
902 * instance, see
903 * {@link #toGregorianCalendar(TimeZone, Locale, XMLGregorianCalendar)}.</p>
904 *
905 * <table border="2" rules="all" cellpadding="2">
906 *   <thead>
907 *     <tr>
908 *       <th align="center" colspan="2">
909 *         Field by Field Conversion from this class to
910 *         <code>java.util.GregorianCalendar</code>
911 *       </th>
912 *     </tr>
913 *   </thead>
914 *   <tbody>
915 *     <tr>
916 *       <td><code>java.util.GregorianCalendar</code> field</td>
917 *       <td><code>javax.xml.datatype.XMLGregorianCalendar</code> field</td>
918 *     </tr>
919 *     <tr>
920 *       <td><code>ERA</code></td>
921 *       <td>{@link #getEonAndYear()}<code>.signum() < 0 ? GregorianCalendar.BC :
GregorianCalendar.AD</code></td>
922 *     </tr>
923 *     <tr>
924 *       <td><code>YEAR</code></td>
925 *       <td>{@link #getEonAndYear()}<code>.abs().intValue()</code><i>*</i></td>
926 *     </tr>
927 *     <tr>
928 *       <td><code>MONTH</code></td>
929 *       <td>{@link #getMonth()} - {@link DatatypeConstants#JANUARY} + {@link GregorianCalendar
#JANUARY}</td>
930 *     </tr>
931 *     <tr>
932 *       <td><code>DAY_OF_MONTH</code></td>
933 *       <td>{@link #getDay()}</td>
934 *     </tr>
935 *     <tr>
936 *       <td><code>HOUR_OF_DAY</code></td>
937 *       <td>{@link #getHour()}</td>
938 *     </tr>
939 *     <tr>
940 *       <td><code>MINUTE</code></td>
941 *       <td>{@link #getMinute()}</td>
942 *     </tr>
943 *     <tr>
944 *       <td><code>SECOND</code></td>
945 *       <td>{@link #getSecond()}</td>
946 *     </tr>
947 *     <tr>
948 *       <td><code>MILLISECOND</code></td>
949 *       <td>get millisecond order from {@link #getFractionalSecond()}<i>*</i> </td>
950 *     </tr>
951 *   </tbody>

```

```

952 *      <td><code>GregorianCalendar.setTimeZone(TimeZone)</code></td>
953 *      <td>{@link #getTimeZone()} formatted into Custom timezone id</td>
954 *    </tr>
955 *  </tbody>
956 * </table>
957 * <i>*</i> designates possible loss of precision during the conversion due
958 * to source datatype having higher precision than target datatype.
959 *
960 * <p>To ensure consistency in conversion implementations, the new
961 * <code>GregorianCalendar</code> should be instantiated in following
962 * manner.
963 * <ul>
964 *   <li>Using <code>timeZone</code> value as defined above, create a new
965 *   <code>java.util.GregorianCalendar(timeZone,Locale.getDefault())</code>.
966 *   </li>
967 *   <li>Initialize all GregorianCalendar fields by calling {@link java.util.GregorianCalendar#
968 *   clear()}.</li>
969 *   <li>Obtain a pure GregorianCalendar by invoking
970 *   <code>GregorianCalendar.setGregorianChange(
971 *   new Date(Long.MIN_VALUE))</code>.</li>
972 *   <li>Its fields ERA, YEAR, MONTH, DAY_OF_MONTH, HOUR_OF_DAY,
973 *   MINUTE, SECOND and MILLISECOND are set using the method
974 *   <code>Calendar.set(int,int)</code></li>
975 * </ul>
976 * </p>
977 * @see #toGregorianCalendar(java.util.TimeZone, java.util.Locale, XMLGregorianCalendar)
978 */
979 public abstract GregorianCalendar toGregorianCalendar();
980
981 /**
982 * <p>Convert this <code>XMLGregorianCalendar</code> along with provided parameters
983 * to a {@link GregorianCalendar} instance.</p>
984 *
985 * <p>Since XML Schema 1.0 date/time datatypes has no concept of
986 * timezone ids or daylight savings timezone ids, this conversion operation
987 * allows the user to explicitly specify one with
988 * <code>timezone</code> parameter.</p>
989 *
990 * <p>To compute the return value's <code>TimeZone</code> field,
991 * <ul>
992 *   <li>when parameter <code>timeZone</code> is non-null,
993 *   it is the timezone field.</li>
994 *   <li>else when <code>this.getTimeZone() != FIELD_UNDEFINED</code>,
995 *   create a <code>java.util.TimeZone</code> with a custom timezone id
996 *   using the <code>this.getTimeZone()</code>.</li>
997 *   <li>else when <code>defaults.getTimeZone() != FIELD_UNDEFINED</code>,
998 *   create a <code>java.util.TimeZone</code> with a custom timezone id
999 *   using <code>defaults.getTimeZone()</code>.</li>
1000 *   <li>else use the <code>GregorianCalendar</code> default timezone value
1001 *   for the host is defined as specified by
1002 *   <code>java.util.TimeZone.getDefault()</code>.</li></ul>
1003 *

```

```

1004 * <p>To ensure consistency in conversion implementations, the new
1005 * <code>GregorianCalendar</code> should be instantiated in following
1006 * manner.
1007 * <ul>
1008 * <li>Create a new <code>java.util.GregorianCalendar(TimeZone,
1009 * <code>Locale)</code> with <code>TimeZone</code> set as specified above and the
1010 * <code>Locale</code> parameter.
1011 * </li>
1012 * <li>Initialize all <code>GregorianCalendar</code> fields by calling {@link GregorianCalendar#clear()}</li>
1013 * <li>Obtain a pure Gregorian Calendar by invoking
1014 * <code>GregorianCalendar.setGregorianChange(
1015 * <code>new Date(Long.MIN_VALUE)</code>)</code>.</li>
1016 * <li>Its fields ERA, YEAR, MONTH, DAY_OF_MONTH, HOUR_OF_DAY,
1017 * <code>MINUTE</code>, <code>SECOND</code> and <code>MILLISECOND</code> are set using the method
1018 * <code>Calendar.set(int,int)</code>.</li>
1019 * </ul>
1020 *
1021 * @param timezone provide Timezone. <code>null</code> is a legal value.
1022 * @param aLocale provide explicit Locale. Use default GregorianCalendar locale if
1023 * value is <code>null</code>.
1024 * @param defaults provide default field values to use when corresponding
1025 * field for this instance is FIELD_UNDEFINED or null.
1026 * If <code>defaults</code> is <code>null</code> or a field
1027 * within the specified <code>defaults</code> is undefined,
1028 * just use <code>java.util.GregorianCalendar</code> defaults.
1029 * @return a java.util.GregorianCalendar conversion of this instance.
1030 */
1031 public abstract GregorianCalendar toGregorianCalendar(
1032     java.util.TimeZone timezone,
1033     java.util.Locale aLocale,
1034     XMLGregorianCalendar defaults);
1035
1036 /**
1037 * <p>Returns a <code>java.util.TimeZone</code> for this class.</p>
1038 *
1039 * <p>If timezone field is defined for this instance,
1040 * returns TimeZone initialized with custom timezone id
1041 * of zoneoffset. If timezone field is undefined,
1042 * try the defaultZoneoffset that was passed in.
1043 * If defaultZoneoffset is FIELD_UNDEFINED, return
1044 * default timezone for this host.
1045 * (Same default as java.util.GregorianCalendar).</p>
1046 *
1047 * @param defaultZoneoffset default zoneoffset if this zoneoffset is
1048 * {@link DatatypeConstants#FIELD_UNDEFINED}.
1049 *
1050 * @return TimeZone for this.
1051 */
1052 public abstract TimeZone getTimeZone(int defaultZoneoffset);
1053
1054
1055

```

```

1056  /**
1057   * <p>Creates and returns a copy of this object.</p>
1058   *
1059   * @return copy of this <code>Object</code>
1060   */
1061  public abstract Object clone();
1062  }

```

## 15 Xml Namespace (javax.xml.namespace package)

### 15.1 NamespaceContext.java

Secuencia 152: javax.xml.namespace.NamespaceContext.java

```

1  /*
2   * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.namespace;
27
28  import java.util.Iterator;
29
30  /**
31   * <p>Interface for read only XML Namespace context processing.</p>
32   *
33   * <p>An XML Namespace has the properties:</p>
34   * <ul>
35   *   <li>Namespace URI:
36   *     Namespace name expressed as a URI to which the prefix is bound</li>
37   *   <li>prefix: syntactically, this is the part of the attribute name
38   *     following the <code>XMLConstants.XMLNS_ATTRIBUTE</code>
39   *     ("xmlns") in the Namespace declaration</li>
40   * </ul>

```

```

41 * <p>example:
42 * <code>&lt;element xmlns:prefix="http://Namespace-name-URI"&gt;</code></p>
43 *
44 * <p>All <code>get*(*)</code> methods operate in the current scope
45 * for Namespace URI and prefix resolution.</p>
46 *
47 * <p>Note that a Namespace URI can be bound to
48 * <strong>multiple</strong> prefixes in the current scope. This can
49 * occur when multiple <code>XMLConstants.XMLNS_ATTRIBUTE</code>
50 * ("xmlns") Namespace declarations occur in the same Start-Tag and
51 * refer to the same Namespace URI. e.g.<br />
52 * <pre>
53 * &lt;element xmlns:prefix1="http://Namespace-name-URI"
54 *           xmlns:prefix2="http://Namespace-name-URI"&gt;
55 * </pre>
56 * This can also occur when the same Namespace URI is used in multiple
57 * <code>XMLConstants.XMLNS_ATTRIBUTE</code> ("xmlns") Namespace
58 * declarations in the logical parent element hierarchy. e.g.<br />
59 * <pre>
60 * &lt;parent xmlns:prefix1="http://Namespace-name-URI">
61 *   &lt;child xmlns:prefix2="http://Namespace-name-URI"&gt;
62 *     ...
63 *   &lt;/child&gt;
64 * &lt;/parent&gt;
65 * </pre></p>
66 *
67 * <p>A prefix can only be bound to a <strong>single</strong>
68 * Namespace URI in the current scope.</p>
69 *
70 * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
71 * @see javax.xml.XMLConstants
72 *   javax.xml.XMLConstants for declarations of common XML values
73 * @see <a href="http://www.w3.org/TR/xmlschema-2/#QName">
74 *   XML Schema Part2: Datatypes</a>
75 * @see <a href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">
76 *   Namespaces in XML</a>
77 * @see <a href="http://www.w3.org/XML/xml-names-19990114-errata">
78 *   Namespaces in XML Errata</a>
79 * @since 1.5
80 */
81
82 public interface NamespaceContext {
83
84     /**
85      * <p>Get Namespace URI bound to a prefix in the current scope.</p>
86      *
87      * <p>When requesting a Namespace URI by prefix, the following
88      * table describes the returned Namespace URI value for all
89      * possible prefix values:</p>
90      *
91      * <table border="2" rules="all" cellpadding="4">
92      *   <thead>
93      *     <tr>

```



```

94      *      <td align="center" colspan="2">
95      *      <code>getNamespaceURI(prefix)</code>
96      *      return value for specified prefixes
97      *      </td>
98      *      </tr>
99      *      <tr>
100     *      <td>prefix parameter</td>
101     *      <td>Namespace URI return value</td>
102     *      </tr>
103     *      </thead>
104     *      <tbody>
105     *      <tr>
106     *      <td><code>DEFAULT_NS_PREFIX</code> ("")</td>
107     *      <td>default Namespace URI in the current scope or
108     *      <code>{@link
109     *      javax.xml.XMLConstants#NULL_NS_URI XMLConstants.NULL_NS_URI("")}
110     *      </code>
111     *      when there is no default Namespace URI in the current scope</td>
112     *      </tr>
113     *      <tr>
114     *      <td>bound prefix</td>
115     *      <td>Namespace URI bound to prefix in current scope</td>
116     *      </tr>
117     *      <tr>
118     *      <td>unbound prefix</td>
119     *      <td>
120     *      <code>{@link
121     *      javax.xml.XMLConstants#NULL_NS_URI XMLConstants.NULL_NS_URI("")}
122     *      </code>
123     *      </td>
124     *      </tr>
125     *      <tr>
126     *      <td><code>XMLConstants.XML_NS_PREFIX</code> ("xml")</td>
127     *      <td><code>XMLConstants.XML_NS_URI</code>
128     *      ("http://www.w3.org/XML/1998/namespace")</td>
129     *      </tr>
130     *      <tr>
131     *      <td><code>XMLConstants.XMLNS_ATTRIBUTE</code> ("xmlns")</td>
132     *      <td><code>XMLConstants.XMLNS_ATTRIBUTE_NS_URI</code>
133     *      ("http://www.w3.org/2000/xmlns/")</td>
134     *      </tr>
135     *      <tr>
136     *      <td><code>null</code></td>
137     *      <td><code>IllegalArgumentException</code> is thrown</td>
138     *      </tr>
139     *      </tbody>
140     *      </table>
141     *
142     *      @param prefix prefix to look up
143     *
144     *      @return Namespace URI bound to prefix in the current scope
145     *
146     *      @throws IllegalArgumentException When <code>prefix</code> is

```

```

147 * <code>null</code>
148 */
149 String getNamespaceURI(String prefix);
150
151 /**
152 * <p>Get prefix bound to Namespace URI in the current scope.</p>
153 *
154 * <p>To get all prefixes bound to a Namespace URI in the current
155 * scope, use {@link #getPrefixes(String namespaceURI)}.</p>
156 *
157 * <p>When requesting a prefix by Namespace URI, the following
158 * table describes the returned prefix value for all Namespace URI
159 * values:</p>
160 *
161 * <table border="2" rules="all" cellpadding="4">
162 *   <thead>
163 *     <tr>
164 *       <th align="center" colspan="2">
165 *         <code>getPrefix(namespaceURI)</code> return value for
166 *         specified Namespace URIs
167 *       </th>
168 *     </tr>
169 *     <tr>
170 *       <th>Namespace URI parameter</th>
171 *       <th>prefix value returned</th>
172 *     </tr>
173 *   </thead>
174 *   <tbody>
175 *     <tr>
176 *       <td>&lt;default Namespace URI&gt;</td>
177 *       <td><code>XMLConstants.DEFAULT_NS_PREFIX</code> ("")
178 *     </td>
179 *     </tr>
180 *     <tr>
181 *       <td>bound Namespace URI</td>
182 *       <td>prefix bound to Namespace URI in the current scope,
183 *       if multiple prefixes are bound to the Namespace URI in
184 *       the current scope, a single arbitrary prefix, whose
185 *       choice is implementation dependent, is returned</td>
186 *     </tr>
187 *     <tr>
188 *       <td>unbound Namespace URI</td>
189 *       <td><code>null</code></td>
190 *     </tr>
191 *     <tr>
192 *       <td><code>XMLConstants.XML_NS_URI</code>
193 *       ("http://www.w3.org/XML/1998/namespace")</td>
194 *       <td><code>XMLConstants.XML_NS_PREFIX</code> ("xml")</td>
195 *     </tr>
196 *     <tr>
197 *       <td><code>XMLConstants.XMLNS_ATTRIBUTE_NS_URI</code>
198 *       ("http://www.w3.org/2000/xmlns/")</td>
199 *       <td><code>XMLConstants.XMLNS_ATTRIBUTE</code> ("xmlns")</td>

```

```

200 *     </tr>
201 *     <tr>
202 *         <td><code>null</code></td>
203 *         <td><code>IllegalArgumentException</code> is thrown</td>
204 *     </tr>
205 * </tbody>
206 * </table>
207 *
208 * @param namespaceURI URI of Namespace to lookup
209 *
210 * @return prefix bound to Namespace URI in current context
211 *
212 * @throws IllegalArgumentException When <code>namespaceURI</code> is
213 *     <code>null</code>
214 */
215 String getPrefix(String namespaceURI);
216
217 /**
218 * <p>Get all prefixes bound to a Namespace URI in the current
219 * scope.</p>
220 *
221 * <p>An Iterator over String elements is returned in an arbitrary,
222 * <strong>implementation dependent</strong>, order.</p>
223 *
224 * <p><strong>The <code>Iterator</code> is
225 * <em>not</em> modifiable. e.g. the
226 * <code>remove()</code> method will throw
227 * <code>UnsupportedOperationException</code>.</strong></p>
228 *
229 * <p>When requesting prefixes by Namespace URI, the following
230 * table describes the returned prefixes value for all Namespace
231 * URI values:</p>
232 *
233 * <table border="2" rules="all" cellpadding="4">
234 *     <thead>
235 *         <tr>
236 *             <th align="center" colspan="2"><code>
237 *                 getPrefixes(namespaceURI)</code> return value for
238 *                 specified Namespace URIs</th>
239 *         </tr>
240 *         <tr>
241 *             <th>Namespace URI parameter</th>
242 *             <th>prefixes value returned</th>
243 *         </tr>
244 *     </thead>
245 *     <tbody>
246 *         <tr>
247 *             <td>bound Namespace URI,
248 *                 including the &lt;default Namespace URI></td>
249 *             <td>
250 *                 <code>Iterator</code> over prefixes bound to Namespace URI in
251 *                 the current scope in an arbitrary,
252 *                 <strong>implementation dependent</strong>,

```

```

253 *      order
254 *      </td>
255 *      </tr>
256 *      <tr>
257 *          <td>unbound Namespace URI</td>
258 *          <td>empty <code>Iterator</code></td>
259 *      </tr>
260 *      <tr>
261 *          <td><code>XMLConstants.XML_NS_URI</code>
262 *              ("http://www.w3.org/XML/1998/namespace")</td>
263 *          <td><code>Iterator</code> with one element set to
264 *              <code>XMLConstants.XML_NS_PREFIX</code> ("xml")</td>
265 *      </tr>
266 *      <tr>
267 *          <td><code>XMLConstants.XMLNS_ATTRIBUTE_NS_URI</code>
268 *              ("http://www.w3.org/2000/xmlns/")</td>
269 *          <td><code>Iterator</code> with one element set to
270 *              <code>XMLConstants.XMLNS_ATTRIBUTE</code> ("xmlns")</td>
271 *      </tr>
272 *      <tr>
273 *          <td><code>null</code></td>
274 *          <td><code>IllegalArgumentException</code> is thrown</td>
275 *      </tr>
276 *      </tbody>
277 *  </table>
278 *
279 *  @param namespaceURI URI of Namespace to lookup
280 *
281 *  @return <code>Iterator</code> for all prefixes bound to the
282 *      Namespace URI in the current scope
283 *
284 *  @throws IllegalArgumentException When <code>namespaceURI</code> is
285 *      <code>null</code>
286 */
287 Iterator getPrefixes(String namespaceURI);
288 }

```

## 15.2 QName.java

Secuencia 153: javax/xml/namespace/QName.java

```

1  /*
2  * Copyright (c) 2003, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```

14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.namespace;
27
28  import java.io.Serializable;
29  import java.security.AccessController;
30  import java.security.PrivilegedAction;
31
32  import javax.xml.XMLConstants;
33
34  /**
35   * <p><code>QName</code> represents a <strong>qualified name</strong>
36   * as defined in the XML specifications: <a
37   * href="http://www.w3.org/TR/xmlschema-2/#QName">XML Schema Part2:
38   * Datatypes specification</a>, <a
39   * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">Namespaces
40   * in XML</a>, <a
41   * href="http://www.w3.org/XML/xml-names-19990114-errata">Namespaces
42   * in XML Errata</a>.</p>
43   *
44   * <p>The value of a <code>QName</code> contains a <strong>Namespace
45   * URI</strong>, <strong>local part</strong> and
46   * <strong>prefix</strong>.</p>
47   *
48   * <p>The prefix is included in <code>QName</code> to retain lexical
49   * information <strong><em>when present</em></strong> in an {@link
50   * javax.xml.transform.Source XML input source}. The prefix is
51   * <strong><em>NOT</em></strong> used in {@link #equals(Object)
52   * QName.equals(Object)} or to compute the {@link #hashCode()
53   * QName.hashCode()}. Equality and the hash code are defined using
54   * <strong><em>only</em></strong> the Namespace URI and local part.</p>
55   *
56   * <p>If not specified, the Namespace URI is set to {@link
57   * javax.xml.XMLConstants#NULL_NS_URI XMLConstants.NULL_NS_URI}.
58   * If not specified, the prefix is set to {@link
59   * javax.xml.XMLConstants#DEFAULT_NS_PREFIX
60   * XMLConstants.DEFAULT_NS_PREFIX}.</p>
61   *
62   * <p><code>QName</code> is immutable.</p>
63   *
64   * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
65   * @version $Revision: 1.8 $, $Date: 2010/03/18 03:06:17 $
66   * @see <a href="http://www.w3.org/TR/xmlschema-2/#QName">

```

```

67 * XML Schema Part2: Datatypes specification</a>
68 * @see <a href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">
69 * Namespaces in XML</a>
70 * @see <a href="http://www.w3.org/XML/xml-names-19990114-errata">
71 * Namespaces in XML Errata</a>
72 * @since 1.5
73 */
74
75 public class QName implements Serializable {
76
77     /**
78      * <p>Stream Unique Identifier.</p>
79      *
80      * <p>Due to a historical defect, QName was released with multiple
81      * serialVersionUID values even though its serialization was the
82      * same.</p>
83      *
84      * <p>To workaroud this issue, serialVersionUID is set with either
85      * a default value or a compatibility value. To use the
86      * compatiblity value, set the system property:</p>
87      *
88      * <code>com.sun.xml.namespace.QName.useCompatibleSerialVersionUID=1.0</code>
89      *
90      * <p>This workaround was inspired by classes in the javax.management
91      * package, e.g. ObjectName, etc.
92      * See CR6267224 for original defect report.</p>
93      */
94     private static final long serialVersionUID;
95     /**
96      * <p>Default <code>serialVersionUID</code> value.</p>
97      */
98     private static final long defaultSerialVersionUID = -9120448754896609940L;
99     /**
100      * <p>Compatibility <code>serialVersionUID</code> value.</p>
101      */
102     private static final long compatibleSerialVersionUID = 4418622981026545151L;
103     /**
104      * <p>Flag to use default or campatible serialVersionUID.</p>
105      */
106     private static boolean useDefaultSerialVersionUID = true;
107     static {
108         try {
109             // use a privileged block as reading a system property
110             String valueUseCompatibleSerialVersionUID = (String) AccessController.doPrivileged(
111                 new PrivilegedAction() {
112                     public Object run() {
113                         return System.getProperty("com.sun.xml.namespace.QName.
114                             useCompatibleSerialVersionUID");
115                     }
116                 });
117             useDefaultSerialVersionUID = (valueUseCompatibleSerialVersionUID != null &&
118                 valueUseCompatibleSerialVersionUID.equals("1.0")) ? false : true;

```

```

118     } catch (Exception exception) {
119         // use default if any Exceptions
120         useDefaultSerialVersionUID = true;
121     }
122
123     // set serialVersionUID to desired value
124     if (useDefaultSerialVersionUID) {
125         serialVersionUID = defaultSerialVersionUID;
126     } else {
127         serialVersionUID = compatibleSerialVersionUID;
128     }
129 }
130
131 /**
132  * <p>Namespace URI of this <code>QName</code>.</p>
133  */
134 private final String namespaceURI;
135
136 /**
137  * <p>local part of this <code>QName</code>.</p>
138  */
139 private final String localPart;
140
141 /**
142  * <p>prefix of this <code>QName</code>.</p>
143  */
144 private final String prefix;
145
146 /**
147  * <p><code>QName</code> constructor specifying the Namespace URI
148  * and local part.</p>
149  *
150  * <p>If the Namespace URI is <code>null</code>, it is set to
151  * {@link javax.xml.XMLConstants#NULL_NS_URI
152  * XMLConstants.NULL_NS_URI}. This value represents no
153  * explicitly defined Namespace as defined by the <a
154  * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">Namespaces
155  * in XML</a> specification. This action preserves compatible
156  * behavior with QName 1.0. Explicitly providing the {@link
157  * javax.xml.XMLConstants#NULL_NS_URI
158  * XMLConstants.NULL_NS_URI} value is the preferred coding
159  * style.</p>
160  *
161  * <p>If the local part is <code>null</code> an
162  * <code>IllegalArgumentException</code> is thrown.
163  * A local part of "" is allowed to preserve
164  * compatible behavior with QName 1.0. </p>
165  *
166  * <p>When using this constructor, the prefix is set to {@link
167  * javax.xml.XMLConstants#DEFAULT_NS_PREFIX
168  * XMLConstants.DEFAULT_NS_PREFIX}.</p>
169  *
170  * <p>The Namespace URI is not validated as a

```

```

171 * <a href="http://www.ietf.org/rfc/rfc2396.txt">URI reference</a>.
172 * The local part is not validated as a
173 * <a href="http://www.w3.org/TR/REC-xml-names/#NT-NCName">NCName</a>
174 * as specified in <a href="http://www.w3.org/TR/REC-xml-names/">Namespaces
175 * in XML</a>.</p>
176 *
177 * @param namespaceURI Namespace URI of the <code>QName</code>
178 * @param localPart local part of the <code>QName</code>
179 *
180 * @throws IllegalArgumentException When <code>localPart</code> is
181 * <code>null</code>
182 *
183 * @see #QName(String namespaceURI, String localPart, String
184 * prefix) QName(String namespaceURI, String localPart, String
185 * prefix)
186 */
187 public QName(final String namespaceURI, final String localPart) {
188     this(namespaceURI, localPart, XMLConstants.DEFAULT_NS_PREFIX);
189 }
190
191 /**
192 * <p><code>QName</code> constructor specifying the Namespace URI,
193 * local part and prefix.</p>
194 *
195 * <p>If the Namespace URI is <code>null</code>, it is set to
196 * {@link javax.xml.XMLConstants#NULL_NS_URI
197 * XMLConstants.NULL_NS_URI}. This value represents no
198 * explicitly defined Namespace as defined by the <a
199 * href="http://www.w3.org/TR/REC-xml-names/#ns-qualnames">Namespaces
200 * in XML</a> specification. This action preserves compatible
201 * behavior with QName 1.0. Explicitly providing the {@link
202 * javax.xml.XMLConstants#NULL_NS_URI
203 * XMLConstants.NULL_NS_URI} value is the preferred coding
204 * style.</p>
205 *
206 * <p>If the local part is <code>null</code> an
207 * <code>IllegalArgumentException</code> is thrown.
208 * A local part of "" is allowed to preserve
209 * compatible behavior with QName 1.0. </p>
210 *
211 * <p>If the prefix is <code>null</code>, an
212 * <code>IllegalArgumentException</code> is thrown. Use {@link
213 * javax.xml.XMLConstants#DEFAULT_NS_PREFIX
214 * XMLConstants.DEFAULT_NS_PREFIX} to explicitly indicate that no
215 * prefix is present or the prefix is not relevant.</p>
216 *
217 * <p>The Namespace URI is not validated as a
218 * <a href="http://www.ietf.org/rfc/rfc2396.txt">URI reference</a>.
219 * The local part and prefix are not validated as a
220 * <a href="http://www.w3.org/TR/REC-xml-names/#NT-NCName">NCName</a>
221 * as specified in <a href="http://www.w3.org/TR/REC-xml-names/">Namespaces
222 * in XML</a>.</p>
223 *

```



```

224 * @param namespaceURI Namespace URI of the <code>QName</code>
225 * @param localPart    local part of the <code>QName</code>
226 * @param prefix       prefix of the <code>QName</code>
227 *
228 * @throws IllegalArgumentException When <code>localPart</code>
229 *   or <code>prefix</code> is <code>null</code>
230 */
231 public QName(String namespaceURI, String localPart, String prefix) {
232
233     // map null Namespace URI to default
234     // to preserve compatibility with QName 1.0
235     if (namespaceURI == null) {
236         this.namespaceURI = XMLConstants.NULL_NS_URI;
237     } else {
238         this.namespaceURI = namespaceURI;
239     }
240
241     // local part is required.
242     // "" is allowed to preserve compatibility with QName 1.0
243     if (localPart == null) {
244         throw new IllegalArgumentException(
245             "local part cannot be \"null\" when creating a QName");
246     }
247     this.localPart = localPart;
248
249     // prefix is required
250     if (prefix == null) {
251         throw new IllegalArgumentException(
252             "prefix cannot be \"null\" when creating a QName");
253     }
254     this.prefix = prefix;
255 }
256
257 /**
258 * <p><code>QName</code> constructor specifying the local part.</p>
259 *
260 * <p>If the local part is <code>null</code> an
261 * <code>IllegalArgumentException</code> is thrown.
262 * A local part of "" is allowed to preserve
263 * compatible behavior with QName 1.0. </p>
264 *
265 * <p>When using this constructor, the Namespace URI is set to
266 * {@link javax.xml.XMLConstants#NULL_NS_URI
267 * XMLConstants.NULL_NS_URI} and the prefix is set to {@link
268 * javax.xml.XMLConstants#DEFAULT_NS_PREFIX
269 * XMLConstants.DEFAULT_NS_PREFIX}.</p>
270 *
271 * <p><em>In an XML context, all Element and Attribute names exist
272 * in the context of a Namespace. Making this explicit during the
273 * construction of a <code>QName</code> helps prevent hard to
274 * diagnosis XML validity errors. The constructors {@link
275 * #QName(String namespaceURI, String localPart) QName(String
276 * namespaceURI, String localPart)} and

```

```

277 * {@link #QName(String namespaceURI, String localPart, String prefix)}
278 * are preferred.</em></p>
279 *
280 * <p>The local part is not validated as a
281 * <a href="http://www.w3.org/TR/REC-xml-names/#NT-NCName">NCName</a>
282 * as specified in <a href="http://www.w3.org/TR/REC-xml-names/">Namespaces
283 * in XML</a>.</p>
284 *
285 * @param localPart local part of the <code>QName</code>
286 *
287 * @throws IllegalArgumentException When <code>localPart</code> is
288 * <code>null</code>
289 *
290 * @see #QName(String namespaceURI, String localPart) QName(String
291 * namespaceURI, String localPart)
292 * @see #QName(String namespaceURI, String localPart, String
293 * prefix) QName(String namespaceURI, String localPart, String
294 * prefix)
295 */
296 public QName(String localPart) {
297     this(
298         XMLConstants.NULL_NS_URI,
299         localPart,
300         XMLConstants.DEFAULT_NS_PREFIX);
301 }
302
303 /**
304 * <p>Get the Namespace URI of this <code>QName</code>.</p>
305 *
306 * @return Namespace URI of this <code>QName</code>
307 */
308 public String getNamespaceURI() {
309     return namespaceURI;
310 }
311
312 /**
313 * <p>Get the local part of this <code>QName</code>.</p>
314 *
315 * @return local part of this <code>QName</code>
316 */
317 public String getLocalPart() {
318     return localPart;
319 }
320
321 /**
322 * <p>Get the prefix of this <code>QName</code>.</p>
323 *
324 * <p>The prefix assigned to a <code>QName</code> might
325 * <strong><em>NOT</em></strong> be valid in a different
326 * context. For example, a <code>QName</code> may be assigned a
327 * prefix in the context of parsing a document but that prefix may
328 * be invalid in the context of a different document.</p>
329 *

```

```

330     * @return prefix of this <code>QName</code>
331     */
332     public String getPrefix() {
333         return prefix;
334     }
335
336     /**
337     * <p>Test this <code>QName</code> for equality with another
338     * <code>Object</code>.</p>
339     *
340     * <p>If the <code>Object</code> to be tested is not a
341     * <code>QName</code> or is <code>null</code>, then this method
342     * returns <code>false</code>.</p>
343     *
344     * <p>Two <code>QName</code>s are considered equal if and only if
345     * both the Namespace URI and local part are equal. This method
346     * uses <code>String.equals()</code> to check equality of the
347     * Namespace URI and local part. The prefix is
348     * <strong><em>NOT</em></strong> used to determine equality.</p>
349     *
350     * <p>This method satisfies the general contract of {@link
351     * java.lang.Object#equals(Object) Object.equals(Object)}</p>
352     *
353     * @param objectToTest the <code>Object</code> to test for
354     * equality with this <code>QName</code>
355     * @return <code>true</code> if the given <code>Object</code> is
356     * equal to this <code>QName</code> else <code>false</code>
357     */
358     public final boolean equals(Object objectToTest) {
359         if (objectToTest == this) {
360             return true;
361         }
362
363         if (objectToTest == null || !(objectToTest instanceof QName)) {
364             return false;
365         }
366
367         QName qName = (QName) objectToTest;
368
369         return localPart.equals(qName.localPart)
370             && namespaceURI.equals(qName.namespaceURI);
371     }
372
373     /**
374     * <p>Generate the hash code for this <code>QName</code>.</p>
375     *
376     * <p>The hash code is calculated using both the Namespace URI and
377     * the local part of the <code>QName</code>. The prefix is
378     * <strong><em>NOT</em></strong> used to calculate the hash
379     * code.</p>
380     *
381     * <p>This method satisfies the general contract of {@link
382     * java.lang.Object#hashCode() Object.hashCode()}</p>

```

```

383     *
384     * @return hash code for this <code>QName</code> <code>Object</code>
385     */
386     public final int hashCode() {
387         return namespaceURI.hashCode() ^ localPart.hashCode();
388     }
389
390     /**
391     * <p><code>String</code> representation of this
392     * <code>QName</code>.</p>
393     *
394     * <p>The commonly accepted way of representing a <code>QName</code>
395     * as a <code>String</code> was
396     * <a href="http://jclark.com/xml/xmlns.htm">defined</a>
397     * by James Clark. Although this is not a <em>standard</em>
398     * specification, it is in common use, e.g. {@link
399     * javax.xml.transform.Transformer#setParameter(String name, Object value)}.
400     * This implementation represents a <code>QName</code> as:
401     * "{" + Namespace URI + "}" + local part. If the Namespace URI
402     * <code>.equals(XMLConstants.NULL_NS_URI)</code>, only the
403     * local part is returned. An appropriate use of this method is
404     * for debugging or logging for human consumption.</p>
405     *
406     * <p>Note the prefix value is <strong><em>NOT</em></strong>
407     * returned as part of the <code>String</code> representation.</p>
408     *
409     * <p>This method satisfies the general contract of {@link
410     * java.lang.Object#toString() Object.toString()}.</p>
411     *
412     * @return <code>String</code> representation of this <code>QName</code>
413     */
414     public String toString() {
415         if (namespaceURI.equals(XMLConstants.NULL_NS_URI)) {
416             return localPart;
417         } else {
418             return "{" + namespaceURI + "}" + localPart;
419         }
420     }
421
422     /**
423     * <p><code>QName</code> derived from parsing the formatted
424     * <code>String</code>.</p>
425     *
426     * <p>If the <code>String</code> is <code>>null</code> or does not conform to
427     * {@link #toString() QName.toString()} formatting, an
428     * <code>IllegalArgumentException</code> is thrown.</p>
429     *
430     * <p><em>The <code>String</code> <strong>MUST</strong> be in the
431     * form returned by {@link #toString() QName.toString()}.</em></p>
432     *
433     * <p>The commonly accepted way of representing a <code>QName</code>
434     * as a <code>String</code> was
435     * <a href="http://jclark.com/xml/xmlns.htm">defined</a>

```

```

436 * by James Clark. Although this is not a <em>standard</em>
437 * specification, it is in common use, e.g. {@link
438 * javax.xml.transform.Transformer#setParameter(String name, Object value)}.
439 * This implementation parses a <code>String</code> formatted
440 * as: "{" + Namespace URI + "}" + local part. If the Namespace
441 * URI <code>.equals(XMLConstants.NULL_NS_URI)</code>, only the
442 * local part should be provided.</p>
443 *
444 * <p>The prefix value <strong><em>CANNOT</em></strong> be
445 * represented in the <code>String</code> and will be set to
446 * {@link javax.xml.XMLConstants#DEFAULT_NS_PREFIX
447 * XMLConstants.DEFAULT_NS_PREFIX}.</p>
448 *
449 * <p>This method does not do full validation of the resulting
450 * <code>QName</code>.
451 * <p>The Namespace URI is not validated as a
452 * <a href="http://www.ietf.org/rfc/rfc2396.txt">URI reference</a>.
453 * The local part is not validated as a
454 * <a href="http://www.w3.org/TR/REC-xml-names/#NT-NCName">NCName</a>
455 * as specified in
456 * <a href="http://www.w3.org/TR/REC-xml-names/">Namespaces in XML</a>.</p>
457 *
458 * @param qNameAsString <code>String</code> representation
459 * of the <code>QName</code>
460 *
461 * @throws IllegalArgumentException When <code>qNameAsString</code> is
462 * <code>null</code> or malformed
463 *
464 * @return <code>QName</code> corresponding to the given <code>String</code>
465 * @see #toString() QName.toString()
466 */
467 public static QName valueOf(String qNameAsString) {
468
469     // null is not valid
470     if (qNameAsString == null) {
471         throw new IllegalArgumentException(
472             "cannot create QName from \"null\" or \"\" String");
473     }
474
475     // "" local part is valid to preserve compatible behavior with QName 1.0
476     if (qNameAsString.length() == 0) {
477         return new QName(
478             XMLConstants.NULL_NS_URI,
479             qNameAsString,
480             XMLConstants.DEFAULT_NS_PREFIX);
481     }
482
483     // local part only?
484     if (qNameAsString.charAt(0) != '{') {
485         return new QName(
486             XMLConstants.NULL_NS_URI,
487             qNameAsString,
488             XMLConstants.DEFAULT_NS_PREFIX);

```

```

489     }
490
491     // Namespace URI improperly specified?
492     if (qNameAsString.startsWith("{ " + XMLConstants.NULL_NS_URI + "}")) {
493         throw new IllegalArgumentException(
494             "Namespace URI .equals(XMLConstants.NULL_NS_URI), "
495             + ".equals(\" " + XMLConstants.NULL_NS_URI + "\"), "
496             + "only the local part, "
497             + "\" \"
498             + qNameAsString.substring(2 + XMLConstants.NULL_NS_URI.length())
499             + "\", "
500             + "should be provided.");
501     }
502
503     // Namespace URI and local part specified
504     int endOfNamespaceURI = qNameAsString.indexOf('}');
505     if (endOfNamespaceURI == -1) {
506         throw new IllegalArgumentException(
507             "cannot create QName from \" "
508             + qNameAsString
509             + "\", missing closing \"}\"");
510     }
511     return new QName(
512         qNameAsString.substring(1, endOfNamespaceURI),
513         qNameAsString.substring(endOfNamespaceURI + 1),
514         XMLConstants.DEFAULT_NS_PREFIX);
515 }
516 }

```

## 16 Xml Parsers (javax.xml.parsers package)

### 16.1 DocumentBuilder.java

Secuencia 154: javax/xml/parsers/DocumentBuilder.java

```

1  /*
2   * Copyright (c) 2000, 2006, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *

```

```
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.parsers;
27
28 import java.io.File;
29 import java.io.IOException;
30 import java.io.InputStream;
31
32 import javax.xml.validation.Schema;
33
34 import org.w3c.dom.Document;
35 import org.w3c.dom.DOMImplementation;
36
37 import org.xml.sax.EntityResolver;
38 import org.xml.sax.ErrorHandler;
39 import org.xml.sax.InputSource;
40 import org.xml.sax.SAXException;
41
42 /**
43  * Defines the API to obtain DOM Document instances from an XML
44  * document. Using this class, an application programmer can obtain a
45  * {@link Document} from XML.<p>
46  *
47  * An instance of this class can be obtained from the
48  * {@link DocumentBuilderFactory#newDocumentBuilder()} method. Once
49  * an instance of this class is obtained, XML can be parsed from a
50  * variety of input sources. These input sources are InputStreams,
51  * Files, URLs, and SAX InputSources.<p>
52  *
53  * Note that this class reuses several classes from the SAX API. This
54  * does not require that the implementor of the underlying DOM
55  * implementation use a SAX parser to parse XML document into a
56  * <code>Document</code>. It merely requires that the implementation
57  * communicate with the application using these existing APIs.
58  *
59  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
60  */
61
62 public abstract class DocumentBuilder {
63
64     /** Protected constructor */
65     protected DocumentBuilder () {
66     }
67
68     /**
69      * <p>Reset this <code>DocumentBuilder</code> to its original configuration.</p>
70      *
71      *
72      * <p><code>DocumentBuilder</code> is reset to the same state as when it was created with
```

```

73  * {@link DocumentBuilderFactory#newDocumentBuilder()}.
74  * <code>reset()</code> is designed to allow the reuse of existing <code>DocumentBuilder</code>
    s
75  * thus saving resources associated with the creation of new <code>DocumentBuilder</code>s.</p>
76  *
77  * <p>The reset <code>DocumentBuilder</code> is not guaranteed to have the same {@link
    EntityResolver} or {@link ErrorHandler}
78  * <code>Object</code>s, e.g. {@link Object#equals(Object obj)}. It is guaranteed to have a
    functionally equal
79  * <code>EntityResolver</code> and <code>ErrorHandler</code>.</p>
80  *
81  * @throws UnsupportedOperationException When implementation does not
82  *     override this method.
83  *
84  * @since 1.5
85  */
86  public void reset() {
87
88      // implementors should override this method
89      throw new UnsupportedOperationException(
90          "This DocumentBuilder, \"" + this.getClass().getName() + "\", does not support the
            reset functionality."
91          + " Specification \"" + this.getClass().getPackage().getSpecificationTitle() + "\"
            "
92          + " version \"" + this.getClass().getPackage().getSpecificationVersion() + "\"
93          );
94  }
95
96  /**
97  * Parse the content of the given <code>InputStream</code> as an XML
98  * document and return a new DOM {@link Document} object.
99  * An <code>IllegalArgumentException</code> is thrown if the
100  * <code>InputStream</code> is null.
101  *
102  * @param is InputStream containing the content to be parsed.
103  *
104  * @return <code>Document</code> result of parsing the
105  *     <code>InputStream</code>
106  *
107  * @throws IOException If any IO errors occur.
108  * @throws SAXException If any parse errors occur.
109  * @throws IllegalArgumentException When <code>is</code> is <code>null</code>
110  *
111  * @see org.xml.sax.DocumentHandler
112  */
113
114  public Document parse(InputStream is)
115      throws SAXException, IOException {
116      if (is == null) {
117          throw new IllegalArgumentException("InputStream cannot be null");
118      }
119
120      InputSource in = new InputSource(is);

```



```
121     return parse(in);
122 }
123
124 /**
125  * Parse the content of the given InputStream as an
126  * XML document and return a new DOM {@link Document} object.
127  * An IllegalArgumentException is thrown if the
128  * InputStream is null.
129  *
130  * @param is InputStream containing the content to be parsed.
131  * @param systemId Provide a base for resolving relative URIs.
132  *
133  * @return A new DOM Document object.
134  *
135  * @throws IOException If any IO errors occur.
136  * @throws SAXException If any parse errors occur.
137  * @throws IllegalArgumentException When is is null
138  *
139  * @see org.xml.sax.DocumentHandler
140  */
141
142 public Document parse(InputStream is, String systemId)
143     throws SAXException, IOException {
144     if (is == null) {
145         throw new IllegalArgumentException("InputStream cannot be null");
146     }
147
148     InputSource in = new InputSource(is);
149     in.setSystemId(systemId);
150     return parse(in);
151 }
152
153 /**
154  * Parse the content of the given URI as an XML document
155  * and return a new DOM {@link Document} object.
156  * An IllegalArgumentException is thrown if the
157  * URI is null.
158  *
159  * @param uri The location of the content to be parsed.
160  *
161  * @return A new DOM Document object.
162  *
163  * @throws IOException If any IO errors occur.
164  * @throws SAXException If any parse errors occur.
165  * @throws IllegalArgumentException When uri is null
166  *
167  * @see org.xml.sax.DocumentHandler
168  */
169
170 public Document parse(String uri)
171     throws SAXException, IOException {
172     if (uri == null) {
173         throw new IllegalArgumentException("URI cannot be null");
174     }
175 }
```

```

174     }
175
176     InputSource in = new InputSource(uri);
177     return parse(in);
178 }
179
180 /**
181  * Parse the content of the given file as an XML document
182  * and return a new DOM {@link Document} object.
183  * An IllegalArgumentException is thrown if the
184  * File is null null.
185  *
186  * @param f The file containing the XML to parse.
187  *
188  * @throws IOException If any IO errors occur.
189  * @throws SAXException If any parse errors occur.
190  * @throws IllegalArgumentException When f is null
191  *
192  * @see org.xml.sax.DocumentHandler
193  * @return A new DOM Document object.
194  */
195
196 public Document parse(File f) throws SAXException, IOException {
197     if (f == null) {
198         throw new IllegalArgumentException("File cannot be null");
199     }
200
201     //convert file to appropriate URI, f.toURI().toASCIIString()
202     //converts the URI to string as per rule specified in
203     //RFC 2396,
204     InputSource in = new InputSource(f.toURI().toASCIIString());
205     return parse(in);
206 }
207
208 /**
209  * Parse the content of the given input source as an XML document
210  * and return a new DOM {@link Document} object.
211  * An IllegalArgumentException is thrown if the
212  * InputSource is null null.
213  *
214  * @param is InputSource containing the content to be parsed.
215  *
216  * @return A new DOM Document object.
217  *
218  * @throws IOException If any IO errors occur.
219  * @throws SAXException If any parse errors occur.
220  * @throws IllegalArgumentException When is is null
221  *
222  * @see org.xml.sax.DocumentHandler
223  */
224
225 public abstract Document parse(InputSource is)
226     throws SAXException, IOException;

```

```
227
228
229 /**
230  * Indicates whether or not this parser is configured to
231  * understand namespaces.
232  *
233  * @return true if this parser is configured to understand
234  *         namespaces; false otherwise.
235  */
236
237 public abstract boolean isNamespaceAware();
238
239 /**
240  * Indicates whether or not this parser is configured to
241  * validate XML documents.
242  *
243  * @return true if this parser is configured to validate
244  *         XML documents; false otherwise.
245  */
246
247 public abstract boolean isValidating();
248
249 /**
250  * Specify the {@link EntityResolver} to be used to resolve
251  * entities present in the XML document to be parsed. Setting
252  * this to null will result in the underlying
253  * implementation using it's own default implementation and
254  * behavior.
255  *
256  * @param er The EntityResolver to be used to resolve entities
257  *           present in the XML document to be parsed.
258  */
259
260 public abstract void setEntityResolver(EntityResolver er);
261
262 /**
263  * Specify the {@link ErrorHandler} to be used by the parser.
264  * Setting this to null will result in the underlying
265  * implementation using it's own default implementation and
266  * behavior.
267  *
268  * @param eh The ErrorHandler to be used by the parser.
269  */
270
271 public abstract void setErrorHandler(ErrorHandler eh);
272
273 /**
274  * Obtain a new instance of a DOM {@link Document} object
275  * to build a DOM tree with.
276  *
277  * @return A new instance of a DOM Document object.
278  */
279
```

```

280 public abstract Document newDocument();
281
282 /**
283  * Obtain an instance of a {@link DOMImplementation} object.
284  *
285  * @return A new instance of a DOMImplementation.
286  */
287
288 public abstract DOMImplementation getDOMImplementation();
289
290 /** <p>Get current state of canonicalization.</p>
291  *
292  * @return current state canonicalization control
293  */
294 /**
295 public boolean getCanonicalization() {
296     return canonicalState;
297 }
298 */
299
300 /** <p>Get a reference to the the {@link Schema} being used by
301  * the XML processor.</p>
302  *
303  * <p>If no schema is being used, null is returned.</p>
304  *
305  * @return {@link Schema} being used or null
306  * if none in use
307  *
308  * @throws UnsupportedOperationException When implementation does not
309  * override this method
310  *
311  * @since 1.5
312  */
313 public Schema getSchema() {
314     throw new UnsupportedOperationException(
315         "This parser does not support specification \""
316         + this.getClass().getPackage().getSpecificationTitle()
317         + "\" version \""
318         + this.getClass().getPackage().getSpecificationVersion()
319         + "\""
320     );
321 }
322
323
324 /**
325  * <p>Get the XInclude processing mode for this parser.</p>
326  *
327  * @return
328  * the return value of
329  * the {@link DocumentBuilderFactory#isXIncludeAware()}
330  * when this parser was created from factory.
331  *
332  * @throws UnsupportedOperationException When implementation does not

```

```

333     *   override this method
334     *
335     *   @since 1.5
336     *
337     *   @see DocumentBuilderFactory#setXIncludeAware(boolean)
338     */
339     public boolean isXIncludeAware() {
340         throw new UnsupportedOperationException(
341             "This parser does not support specification \""
342             + this.getClass().getPackage().getSpecificationTitle()
343             + "\" version \""
344             + this.getClass().getPackage().getSpecificationVersion()
345             + "\""
346         );
347     }
348 }

```

## 16.2 DocumentBuilderFactory.java

Secuencia 155: javax/xml/parsers/DocumentBuilderFactory.java

```

1  /*
2  * Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.parsers;
27
28 import javax.xml.validation.Schema;
29
30 /**
31  * Defines a factory API that enables applications to obtain a
32  * parser that produces DOM object trees from XML documents.
33  *

```

```

34 * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
35 * @author <a href="mailto:Neeraj.Bajaj@sun.com">Neeraj Bajaj</a>
36 *
37 * @version $Revision: 1.9 $, $Date: 2010/05/25 16:19:44 $
38
39 */
40
41 public abstract class DocumentBuilderFactory {
42
43     private boolean validating = false;
44     private boolean namespaceAware = false;
45     private boolean whitespace = false;
46     private boolean expandEntityRef = true;
47     private boolean ignoreComments = false;
48     private boolean coalescing = false;
49
50     /**
51      * <p>Protected constructor to prevent instantiation.
52      * Use {@link #newInstance()}.</p>
53      */
54     protected DocumentBuilderFactory () {
55     }
56
57     /**
58      * Obtain a new instance of a
59      * <code>DocumentBuilderFactory</code>. This static method creates
60      * a new factory instance.
61      * This method uses the following ordered lookup procedure to determine
62      * the <code>DocumentBuilderFactory</code> implementation class to
63      * load:
64      * <ul>
65      * <li>
66      * Use the <code>javax.xml.parsers.DocumentBuilderFactory</code> system
67      * property.
68      * </li>
69      * <li>
70      * Use the properties file "lib/jaxp.properties" in the JRE directory.
71      * This configuration file is in standard <code>java.util.Properties
72      * </code> format and contains the fully qualified name of the
73      * implementation class with the key being the system property defined
74      * above.
75      *
76      * The jaxp.properties file is read only once by the JAXP implementation
77      * and it's values are then cached for future use. If the file does not exist
78      * when the first attempt is made to read from it, no further attempts are
79      * made to check for its existence. It is not possible to change the value
80      * of any property in jaxp.properties after it has been read for the first time.
81      * </li>
82      * <li>
83      * Uses the service-provider loading facilities, defined by the
84      * {@link java.util.ServiceLoader} class, to attempt to locate and load an
85      * implementation of the service using the {@link plain
86      * java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:

```

```

87      * the service-provider loading facility will use the {@linkplain
88      * java.lang.Thread#getContextClassLoader() current thread's context class loader}
89      * to attempt to load the service. If the context class
90      * loader is null, the {@linkplain
91      * ClassLoader#getSystemClassLoader() system class loader} will be used.
92      * </li>
93      * <li>
94      * Otherwise, the system-default implementation is returned.
95      * </li>
96      * </ul>
97      *
98      * Once an application has obtained a reference to a
99      * <code>DocumentBuilderFactory</code> it can use the factory to
100     * configure and obtain parser instances.
101     *
102     *
103     * <h2>Tip for Trouble-shooting</h2>
104     * <p>Setting the <code>jaxp.debug</code> system property will cause
105     * this method to print a lot of debug messages
106     * to <code>System.err</code> about what it is doing and where it is looking at.</p>
107     *
108     * <p> If you have problems loading {@link DocumentBuilder}s, try:</p>
109     * <pre>
110     * java -Djaxp.debug=1 YourProgram ....
111     * </pre>
112     *
113     * @return New instance of a <code>DocumentBuilderFactory</code>
114     *
115     * @throws FactoryConfigurationError in case of {@linkplain
116     * java.util.ServiceConfigurationError service configuration error} or if
117     * the implementation is not available or cannot be instantiated.
118     */
119     public static DocumentBuilderFactory newInstance() {
120         return FactoryFinder.find(
121             /* The default property name according to the JAXP spec */
122             DocumentBuilderFactory.class, // "javax.xml.parsers.DocumentBuilderFactory"
123             /* The fallback implementation class name */
124             "com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl");
125     }
126
127     /**
128     * <p>Obtain a new instance of a <code>DocumentBuilderFactory</code> from class name.
129     * This function is useful when there are multiple providers in the classpath.
130     * It gives more control to the application as it can specify which provider
131     * should be loaded.</p>
132     *
133     * <p>Once an application has obtained a reference to a <code>DocumentBuilderFactory</code>
134     * it can use the factory to configure and obtain parser instances.</p>
135     *
136     *
137     * <h2>Tip for Trouble-shooting</h2>
138     * <p>Setting the <code>jaxp.debug</code> system property will cause
139     * this method to print a lot of debug messages

```

```

140 * to <code>System.err</code> about what it is doing and where it is looking at.</p>
141 *
142 * <p> If you have problems try:</p>
143 * <pre>
144 * java -Djaxp.debug=1 YourProgram ....
145 * </pre>
146 *
147 * @param factoryClassName fully qualified factory class name that provides implementation of <
148 *   <code>javax.xml.parsers.DocumentBuilderFactory</code>.
149 *
150 * @param classLoader <code>ClassLoader</code> used to load the factory class. If <code>null</code>/
151 *   <code></code>
152 *   current <code>Thread</code>'s context classLoader is used to load the
153 *   factory class.
154 *
155 * @return New instance of a <code>DocumentBuilderFactory</code>
156 *
157 * @throws FactoryConfigurationError if <code>factoryClassName</code> is <code>null</code>, or
158 *   the factory class cannot be loaded, instantiated.
159 *
160 * @see #newInstance()
161 *
162 * @since 1.6
163 */
164 public static DocumentBuilderFactory newInstance(String factoryClassName, ClassLoader
165   classLoader){
166     //do not fallback if given classloader can't find the class, throw exception
167     return FactoryFinder.newInstance(DocumentBuilderFactory.class,
168       factoryClassName, classLoader, false);
169 }
170
171 /**
172 * Creates a new instance of a {@link javax.xml.parsers.DocumentBuilder}
173 * using the currently configured parameters.
174 *
175 * @return A new instance of a DocumentBuilder.
176 *
177 * @throws ParserConfigurationException if a DocumentBuilder
178 *   cannot be created which satisfies the configuration requested.
179 */
180
181 public abstract DocumentBuilder newDocumentBuilder()
182   throws ParserConfigurationException;
183
184 /**
185 * Specifies that the parser produced by this code will
186 * provide support for XML namespaces. By default the value of this is set
187 * to <code>false</code>
188 *
189 * @param awareness true if the parser produced will provide support
190 *   for XML namespaces; false otherwise.
191 */

```



```
189
190 public void setNamespaceAware(boolean awareness) {
191     this.namespaceAware = awareness;
192 }
193
194 /**
195  * Specifies that the parser produced by this code will
196  * validate documents as they are parsed. By default the value of this
197  * is set to false.
198  *
199  * <p>
200  * Note that "the validation" here means
201  * <a href="http://www.w3.org/TR/REC-xml#proc-types">a validating
202  * parser</a> as defined in the XML recommendation.
203  * In other words, it essentially just controls the DTD validation.
204  * (except the legacy two properties defined in JAXP 1.2.)
205  * </p>
206  *
207  * <p>
208  * To use modern schema languages such as W3C XML Schema or
209  * RELAX NG instead of DTD, you can configure your parser to be
210  * a non-validating parser by leaving the {@link #setValidating(boolean)}
211  * method false, then use the {@link #setSchema(Schema)}
212  * method to associate a schema to a parser.
213  * </p>
214  *
215  * @param validating true if the parser produced will validate documents
216  *                   as they are parsed; false otherwise.
217  */
218
219 public void setValidating(boolean validating) {
220     this.validating = validating;
221 }
222
223 /**
224  * Specifies that the parsers created by this factory must eliminate
225  * whitespace in element content (sometimes known loosely as
226  * 'ignorable whitespace') when parsing XML documents (see XML Rec
227  * 2.10). Note that only whitespace which is directly contained within
228  * element content that has an element only content model (see XML
229  * Rec 3.2.1) will be eliminated. Due to reliance on the content model
230  * this setting requires the parser to be in validating mode. By default
231  * the value of this is set to false.
232  *
233  * @param whitespace true if the parser created must eliminate whitespace
234  *                   in the element content when parsing XML documents;
235  *                   false otherwise.
236  */
237
238 public void setIgnoringElementContentWhitespace(boolean whitespace) {
239     this.whitespace = whitespace;
240 }
241
```

```
242 /**
243  * Specifies that the parser produced by this code will
244  * expand entity reference nodes. By default the value of this is set to
245  * <code>true</code>
246  *
247  * @param expandEntityRef true if the parser produced will expand entity
248  * reference nodes; false otherwise.
249  */
250
251 public void setExpandEntityReferences(boolean expandEntityRef) {
252     this.expandEntityRef = expandEntityRef;
253 }
254
255 /**
256  * <p>Specifies that the parser produced by this code will
257  * ignore comments. By default the value of this is set to <code>false
258  * </code>.</p>
259  *
260  * @param ignoreComments <code>boolean</code> value to ignore comments during processing
261  */
262
263 public void setIgnoringComments(boolean ignoreComments) {
264     this.ignoreComments = ignoreComments;
265 }
266
267 /**
268  * Specifies that the parser produced by this code will
269  * convert CDATA nodes to Text nodes and append it to the
270  * adjacent (if any) text node. By default the value of this is set to
271  * <code>false</code>
272  *
273  * @param coalescing true if the parser produced will convert CDATA nodes
274  * to Text nodes and append it to the adjacent (if any)
275  * text node; false otherwise.
276  */
277
278 public void setCoalescing(boolean coalescing) {
279     this.coalescing = coalescing;
280 }
281
282 /**
283  * Indicates whether or not the factory is configured to produce
284  * parsers which are namespace aware.
285  *
286  * @return true if the factory is configured to produce parsers which
287  * are namespace aware; false otherwise.
288  */
289
290 public boolean isNamespaceAware() {
291     return namespaceAware;
292 }
293
294 /**
```

```
295     * Indicates whether or not the factory is configured to produce
296     * parsers which validate the XML content during parse.
297     *
298     * @return true if the factory is configured to produce parsers
299     *         which validate the XML content during parse; false otherwise.
300     */
301
302     public boolean isValidating() {
303         return validating;
304     }
305
306     /**
307     * Indicates whether or not the factory is configured to produce
308     * parsers which ignore ignorable whitespace in element content.
309     *
310     * @return true if the factory is configured to produce parsers
311     *         which ignore ignorable whitespace in element content;
312     *         false otherwise.
313     */
314
315     public boolean isIgnoringElementContentWhitespace() {
316         return whitespace;
317     }
318
319     /**
320     * Indicates whether or not the factory is configured to produce
321     * parsers which expand entity reference nodes.
322     *
323     * @return true if the factory is configured to produce parsers
324     *         which expand entity reference nodes; false otherwise.
325     */
326
327     public boolean isExpandEntityReferences() {
328         return expandEntityRef;
329     }
330
331     /**
332     * Indicates whether or not the factory is configured to produce
333     * parsers which ignores comments.
334     *
335     * @return true if the factory is configured to produce parsers
336     *         which ignores comments; false otherwise.
337     */
338
339     public boolean isIgnoringComments() {
340         return ignoreComments;
341     }
342
343     /**
344     * Indicates whether or not the factory is configured to produce
345     * parsers which converts CDATA nodes to Text nodes and appends it to
346     * the adjacent (if any) Text node.
347     *
```

```

348     * @return true if the factory is configured to produce parsers
349     *         which converts CDATA nodes to Text nodes and appends it to
350     *         the adjacent (if any) Text node; false otherwise.
351     */
352
353     public boolean isCoalescing() {
354         return coalescing;
355     }
356
357     /**
358     * Allows the user to set specific attributes on the underlying
359     * implementation.
360     * <p>
361     * All implementations that implement JAXP 1.5 or newer are required to
362     * support the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} and
363     * {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} properties.
364     * </p>
365     * <ul>
366     * <li>
367     * <p>
368     * Setting the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} property
369     * restricts the access to external DTDs, external Entity References to the
370     * protocols specified by the property.
371     * If access is denied during parsing due to the restriction of this property,
372     * {@link org.xml.sax.SAXException} will be thrown by the parse methods defined by
373     * {@link javax.xml.parsers.DocumentBuilder}.
374     * </p>
375     * <p>
376     * Setting the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} property
377     * restricts the access to external Schema set by the schemaLocation attribute to
378     * the protocols specified by the property. If access is denied during parsing
379     * due to the restriction of this property, {@link org.xml.sax.SAXException}
380     * will be thrown by the parse methods defined by
381     * {@link javax.xml.parsers.DocumentBuilder}.
382     * </p>
383     * </li>
384     * </ul>
385     *
386     * @param name The name of the attribute.
387     * @param value The value of the attribute.
388     *
389     * @throws IllegalArgumentException thrown if the underlying
390     *         implementation doesn't recognize the attribute.
391     */
392     public abstract void setAttribute(String name, Object value)
393         throws IllegalArgumentException;
394
395     /**
396     * Allows the user to retrieve specific attributes on the underlying
397     * implementation.
398     *
399     * @param name The name of the attribute.
400     *

```

```

401     * @return value The value of the attribute.
402     *
403     * @throws IllegalArgumentException thrown if the underlying
404     *     implementation doesn't recognize the attribute.
405     */
406     public abstract Object getAttribute(String name)
407         throws IllegalArgumentException;
408
409     /**
410     * <p>Set a feature for this <code>DocumentBuilderFactory</code> and <code>DocumentBuilder</code>s
411     *     created by this factory.</p>
412     *
413     * <p>
414     *     Feature names are fully qualified {@link java.net.URI}s.
415     *     Implementations may define their own features.
416     *     A {@link ParserConfigurationException} is thrown if this <code>DocumentBuilderFactory</code>
417     *     or the
418     *     <code>DocumentBuilder</code>s it creates cannot support the feature.
419     *     It is possible for a <code>DocumentBuilderFactory</code> to expose a feature value but be
420     *     unable to change its state.
421     * </p>
422     *
423     * <p>
424     *     All implementations are required to support the {@link javax.xml.XMLConstants#
425     *     FEATURE_SECURE_PROCESSING} feature.
426     *     When the feature is:</p>
427     * <ul>
428     * <li>
429     *     <code>true</code>: the implementation will limit XML processing to conform to
430     *     implementation limits.
431     *     Examples include entity expansion limits and XML Schema constructs that would consume
432     *     large amounts of resources.
433     *     If XML processing is limited for security reasons, it will be reported via a call to the
434     *     registered
435     *     {@link org.xml.sax.ErrorHandler#fatalError(SAXParseException exception)}.
436     *     See {@link DocumentBuilder#setErrorHandler(org.xml.sax.ErrorHandler errorHandler)}.
437     * </li>
438     * <li>
439     *     <code>false</code>: the implementation will processing XML according to the XML
440     *     specifications without
441     *     regard to possible implementation limits.
442     * </li>
443     * </ul>
444     *
445     * @param name Feature name.
446     * @param value Is feature state <code>true</code> or <code>false</code>.
447     *
448     * @throws ParserConfigurationException if this <code>DocumentBuilderFactory</code> or the <code>DocumentBuilder</code>s
449     *     it creates cannot support this feature.
450     * @throws NullPointerException If the <code>name</code> parameter is null.
451     */
452     public abstract void setFeature(String name, boolean value)

```

```

445         throws ParserConfigurationException;
446
447     /**
448     * <p>Get the state of the named feature.</p>
449     *
450     * <p>
451     * Feature names are fully qualified {@link java.net.URI}s.
452     * Implementations may define their own features.
453     * An {@link ParserConfigurationException} is thrown if this <code>DocumentBuilderFactory</code>
454     * > or the
455     * <code>DocumentBuilder</code>s it creates cannot support the feature.
456     * It is possible for an <code>DocumentBuilderFactory</code> to expose a feature value but be
457     * unable to change its state.
458     * </p>
459     *
460     * @param name Feature name.
461     *
462     * @return State of the named feature.
463     *
464     * @throws ParserConfigurationException if this <code>DocumentBuilderFactory</code>
465     * or the <code>DocumentBuilder</code>s it creates cannot support this feature.
466     */
467     public abstract boolean getFeature(String name)
468         throws ParserConfigurationException;
469
470     /**
471     * Gets the {@link Schema} object specified through
472     * the {@link #setSchema(Schema schema)} method.
473     *
474     * @return
475     * the {@link Schema} object that was last set through
476     * the {@link #setSchema(Schema)} method, or null
477     * if the method was not invoked since a {@link DocumentBuilderFactory}
478     * is created.
479     *
480     * @throws UnsupportedOperationException When implementation does not
481     * override this method.
482     *
483     * @since 1.5
484     */
485     public Schema getSchema() {
486         throw new UnsupportedOperationException(
487             "This parser does not support specification \""
488             + this.getClass().getPackage().getSpecificationTitle()
489             + "\" version \""
490             + this.getClass().getPackage().getSpecificationVersion()
491             + "\"");
492     }
493 }
494
495 /**

```

```
496 * <p>Set the {@link Schema} to be used by parsers created
497 * from this factory.
498 *
499 * <p>
500 * When a {@link Schema} is non-null, a parser will use a validator
501 * created from it to validate documents before it passes information
502 * down to the application.
503 *
504 * <p>When errors are found by the validator, the parser is responsible
505 * to report them to the user-specified {@link org.xml.sax.ErrorHandler}
506 * (or if the error handler is not set, ignore them or throw them), just
507 * like any other errors found by the parser itself.
508 * In other words, if the user-specified {@link org.xml.sax.ErrorHandler}
509 * is set, it must receive those errors, and if not, they must be
510 * treated according to the implementation specific
511 * default error handling rules.
512 *
513 * <p>
514 * A validator may modify the outcome of a parse (for example by
515 * adding default values that were missing in documents), and a parser
516 * is responsible to make sure that the application will receive
517 * modified DOM trees.
518 *
519 * <p>
520 * Initially, null is set as the {@link Schema}.
521 *
522 * <p>
523 * This processing will take effect even if
524 * the {@link #isValidating()} method returns false.
525 *
526 * <p>It is an error to use
527 * the http://java.sun.com/xml/jaxp/properties/schemaSource
528 * property and/or the http://java.sun.com/xml/jaxp/properties/schemaLanguage
529 * property in conjunction with a {@link Schema} object.
530 * Such configuration will cause a {@link ParserConfigurationException}
531 * exception when the {@link #newDocumentBuilder()} is invoked.</p>
532 *
533 *
534 * <h4>Note for implmentors</h4>
535 *
536 * <p>
537 * A parser must be able to work with any {@link Schema}
538 * implementation. However, parsers and schemas are allowed
539 * to use implementation-specific custom mechanisms
540 * as long as they yield the result described in the specification.
541 * </p>
542 *
543 * @param schema Schema to use or null
544 * to remove a schema.
545 *
546 * @throws UnsupportedOperationException When implementation does not
547 * override this method.
548 *
```

```

549     * @since 1.5
550     */
551     public void setSchema(Schema schema) {
552         throw new UnsupportedOperationException(
553             "This parser does not support specification \""
554             + this.getClass().getPackage().getSpecificationTitle()
555             + "\" version \""
556             + this.getClass().getPackage().getSpecificationVersion()
557             + "\"";
558         );
559     }
560
561
562
563     /**
564     * <p>Set state of XInclude processing.</p>
565     *
566     * <p>If XInclude markup is found in the document instance, should it be
567     * processed as specified in <a href="http://www.w3.org/TR/xinclude/">
568     * XML Inclusions (XInclude) Version 1.0</a>.</p>
569     *
570     * <p>XInclude processing defaults to <code>>false</code>.</p>
571     *
572     * @param state Set XInclude processing to <code>>true</code> or
573     * <code>>false</code>
574     *
575     * @throws UnsupportedOperationException When implementation does not
576     * override this method.
577     *
578     * @since 1.5
579     */
580     public void setXIncludeAware(final boolean state) {
581         if (state) {
582             throw new UnsupportedOperationException(" setXIncludeAware " +
583                 "is not supported on this JAXP" +
584                 " implementation or earlier: " + this.getClass());
585         }
586     }
587
588     /**
589     * <p>Get state of XInclude processing.</p>
590     *
591     * @return current state of XInclude processing
592     *
593     * @throws UnsupportedOperationException When implementation does not
594     * override this method.
595     *
596     * @since 1.5
597     */
598     public boolean isXIncludeAware() {
599         throw new UnsupportedOperationException(
600             "This parser does not support specification \""
601             + this.getClass().getPackage().getSpecificationTitle()

```



```

602         + "\" version \""
603         + this.getClass().getPackage().getSpecificationVersion()
604         + "\"
605         );
606     }
607 }

```

### 16.3 FactoryConfigurationError.java

Secuencia 156: javax/xml/parsers/FactoryConfigurationError.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.parsers;
27
28 /**
29  * Thrown when a problem with configuration with the Parser Factories
30  * exists. This error will typically be thrown when the class of a
31  * parser factory specified in the system properties cannot be found
32  * or instantiated.
33  *
34  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
35  * @version $Revision: 1.7 $, $Date: 2010-11-01 04:36:09 $
36  */
37
38 public class FactoryConfigurationError extends Error {
39     private static final long serialVersionUID = -827108682472263355L;
40
41     /**
42      * <code>Exception</code> that represents the error.
43      */

```

```
44 private Exception exception;
45
46 /**
47  * Create a new <code>FactoryConfigurationError</code> with no
48  * detail message.
49  */
50
51 public FactoryConfigurationError() {
52     super();
53     this.exception = null;
54 }
55
56 /**
57  * Create a new <code>FactoryConfigurationError</code> with
58  * the <code>String</code> specified as an error message.
59  *
60  * @param msg The error message for the exception.
61  */
62
63 public FactoryConfigurationError(String msg) {
64     super(msg);
65     this.exception = null;
66 }
67
68
69 /**
70  * Create a new <code>FactoryConfigurationError</code> with a
71  * given <code>Exception</code> base cause of the error.
72  *
73  * @param e The exception to be encapsulated in a
74  * FactoryConfigurationError.
75  */
76
77 public FactoryConfigurationError(Exception e) {
78     super(e.toString());
79     this.exception = e;
80 }
81
82 /**
83  * Create a new <code>FactoryConfigurationError</code> with the
84  * given <code>Exception</code> base cause and detail message.
85  *
86  * @param e The exception to be encapsulated in a
87  * FactoryConfigurationError
88  * @param msg The detail message.
89  */
90
91 public FactoryConfigurationError(Exception e, String msg) {
92     super(msg);
93     this.exception = e;
94 }
95
96
```

```

97  /**
98   * Return the message (if any) for this error . If there is no
99   * message for the exception and there is an encapsulated
100  * exception then the message of that exception, if it exists will be
101  * returned. Else the name of the encapsulated exception will be
102  * returned.
103  *
104  * @return The error message.
105  */
106
107  public String getMessage () {
108      String message = super.getMessage ();
109
110      if (message == null && exception != null) {
111          return exception.getMessage();
112      }
113
114      return message;
115  }
116
117  /**
118   * Return the actual exception (if any) that caused this exception to
119   * be raised.
120   *
121   * @return The encapsulated exception, or null if there is none.
122   */
123
124  public Exception getException () {
125      return exception;
126  }
127
128  /**
129   * use the exception chaining mechanism of JDK1.4
130   */
131  @Override
132  public Throwable getCause() {
133      return exception;
134  }
135  }

```

## 16.4 FactoryFinder.java

Secuencia 157: javax/xml/parsers/FactoryFinder.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```

```
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.parsers;
27
28 import java.io.File;
29 import java.security.AccessController;
30 import java.security.PrivilegedAction;
31 import java.util.Iterator;
32 import java.util.Properties;
33 import java.util.ServiceConfigurationError;
34 import java.util.ServiceLoader;
35
36 /**
37  * <p>Implements pluggable Parsers.</p>
38  *
39  * <p>This class is duplicated for each JAXP subpackage so keep it in
40  * sync. It is package private for secure class loading.</p>
41  *
42  * @author Santiago.PericasGeertsens@sun.com
43  * @author Huizhe.Wang@oracle.com
44  */
45 class FactoryFinder {
46     private static final String DEFAULT_PACKAGE = "com.sun.org.apache.xerces.internal";
47     /**
48      * Internal debug flag.
49      */
50     private static boolean debug = false;
51
52     /**
53      * Cache for properties in java.home/lib/jaxp.properties
54      */
55     private static final Properties cacheProps = new Properties();
56
57     /**
58      * Flag indicating if properties from java.home/lib/jaxp.properties
59      * have been cached.
60      */
61     static volatile boolean firstTime = true;
62
63     /**
```

```
64     * Security support class use to check access control before
65     * getting certain system resources.
66     */
67     private static final SecuritySupport ss = new SecuritySupport();
68
69     // Define system property "jaxp.debug" to get output
70     static {
71         // Use try/catch block to support applets, which throws
72         // SecurityException out of this code.
73         try {
74             String val = ss.getProperty("jaxp.debug");
75             // Allow simply setting the prop to turn on debug
76             debug = val != null && !"false".equals(val);
77         }
78         catch (SecurityException se) {
79             debug = false;
80         }
81     }
82
83     private static void dPrint(String msg) {
84         if (debug) {
85             System.err.println("JAXP: " + msg);
86         }
87     }
88
89     /**
90     * Attempt to load a class using the class loader supplied. If that fails
91     * and fall back is enabled, the current (i.e. bootstrap) class loader is
92     * tried.
93     *
94     * If the class loader supplied is <code>null</code>, first try using the
95     * context class loader followed by the current (i.e. bootstrap) class
96     * loader.
97     *
98     * Use bootstrap classLoader if cl = null and useBSClsLoader is true
99     */
100     static private Class<?> getProviderClass(String className, ClassLoader cl,
101         boolean doFallback, boolean useBSClsLoader) throws ClassNotFoundException
102     {
103         try {
104             if (cl == null) {
105                 if (useBSClsLoader) {
106                     return Class.forName(className, false, FactoryFinder.class.getClassLoader());
107                 } else {
108                     cl = ss.getContextClassLoader();
109                     if (cl == null) {
110                         throw new ClassNotFoundException();
111                     }
112                     else {
113                         return Class.forName(className, false, cl);
114                     }
115                 }
116             }
117         }
```

```

117         else {
118             return Class.forName(className, false, cl);
119         }
120     }
121     catch (ClassNotFoundException e1) {
122         if (doFallback) {
123             // Use current class loader - should always be bootstrap CL
124             return Class.forName(className, false, FactoryFinder.class.getClassLoader());
125         }
126         else {
127             throw e1;
128         }
129     }
130 }
131
132 /**
133  * Create an instance of a class. Delegates to method
134  * <code>getProviderClass()</code> in order to load the class.
135  *
136  * @param type Base class / Service interface of the factory to
137  *             instantiate.
138  *
139  * @param className Name of the concrete class corresponding to the
140  * service provider
141  *
142  * @param cl <code>ClassLoader</code> used to load the factory class. If <code>null</code>
143  * current <code>Thread</code>'s context classLoader is used to load the factory class.
144  *
145  * @param doFallback True if the current ClassLoader should be tried as
146  * a fallback if the class is not found using cl
147  */
148 static <T> T newInstance(Class<T> type, String className, ClassLoader cl,
149                          boolean doFallback)
150     throws FactoryConfigurationError
151 {
152     return newInstance(type, className, cl, doFallback, false);
153 }
154
155 /**
156  * Create an instance of a class. Delegates to method
157  * <code>getProviderClass()</code> in order to load the class.
158  *
159  * @param type Base class / Service interface of the factory to
160  *             instantiate.
161  *
162  * @param className Name of the concrete class corresponding to the
163  * service provider
164  *
165  * @param cl <code>ClassLoader</code> used to load the factory class. If <code>null</code>
166  * current <code>Thread</code>'s context classLoader is used to load the factory class.
167  *
168  * @param doFallback True if the current ClassLoader should be tried as
169  * a fallback if the class is not found using cl

```

```

170     *
171     * @param useBSClsLoader True if cl=null actually meant bootstrap classLoader. This parameter
172     * is needed since DocumentBuilderFactory/SAXParserFactory defined null as context classLoader.
173     */
174     static <T> T newInstance(Class<T> type, String className, ClassLoader cl,
175                             boolean doFallback, boolean useBSClsLoader)
176         throws FactoryConfigurationError
177     {
178         assert type != null;
179         // make sure we have access to restricted packages
180         if (System.getSecurityManager() != null) {
181             if (className != null && className.startsWith(DEFAULT_PACKAGE)) {
182                 cl = null;
183                 useBSClsLoader = true;
184             }
185         }
186
187         try {
188             Class<?> providerClass = getProviderClass(className, cl, doFallback, useBSClsLoader);
189             if (!type.isAssignableFrom(providerClass)) {
190                 throw new ClassCastException(className + " cannot be cast to " + type.getName());
191             }
192             Object instance = providerClass.newInstance();
193             if (debug) { // Extra check to avoid computing cl strings
194                 dPrint("created new instance of " + providerClass +
195                     " using ClassLoader: " + cl);
196             }
197             return type.cast(instance);
198         }
199         catch (ClassNotFoundException x) {
200             throw new FactoryConfigurationError(x,
201                 "Provider " + className + " not found");
202         }
203         catch (Exception x) {
204             throw new FactoryConfigurationError(x,
205                 "Provider " + className + " could not be instantiated: " + x);
206         }
207     }
208
209     /**
210     * Finds the implementation Class object in the specified order. Main
211     * entry point.
212     * @return Class object of factory, never null
213     *
214     * @param type          Base class / Service interface of the
215     *                       factory to find.
216     * @param fallbackClassName Implementation class name, if nothing else
217     *                       is found. Use null to mean no fallback.
218     *
219     * Package private so this code can be shared.
220     */
221     static <T> T find(Class<T> type, String fallbackClassName)
222         throws FactoryConfigurationError

```

```
223 {
224     final String factoryId = type.getName();
225     dPrint("find factoryId =" + factoryId);
226
227     // Use the system property first
228     try {
229         String systemProp = ss.getSystemProperty(factoryId);
230         if (systemProp != null) {
231             dPrint("found system property, value=" + systemProp);
232             return newInstance(type, systemProp, null, true);
233         }
234     }
235     catch (SecurityException se) {
236         if (debug) se.printStackTrace();
237     }
238
239     // try to read from $java.home/lib/jaxp.properties
240     try {
241         if (firstTime) {
242             synchronized (cacheProps) {
243                 if (firstTime) {
244                     String configFile = ss.getSystemProperty("java.home") + File.separator +
245                         "lib" + File.separator + "jaxp.properties";
246                     File f = new File(configFile);
247                     firstTime = false;
248                     if (ss.isFileExists(f)) {
249                         dPrint("Read properties file "+f);
250                         cacheProps.load(ss.getFileInputStream(f));
251                     }
252                 }
253             }
254         }
255         final String factoryClassName = cacheProps.getProperty(factoryId);
256
257         if (factoryClassName != null) {
258             dPrint("found in $java.home/jaxp.properties, value=" + factoryClassName);
259             return newInstance(type, factoryClassName, null, true);
260         }
261     }
262     catch (Exception ex) {
263         if (debug) ex.printStackTrace();
264     }
265
266     // Try Jar Service Provider Mechanism
267     T provider = findServiceProvider(type);
268     if (provider != null) {
269         return provider;
270     }
271     if (fallbackClassName == null) {
272         throw new FactoryConfigurationError(
273             "Provider for " + factoryId + " cannot be found");
274     }
275 }
```



```

276     dPrint("loaded from fallback value: " + fallbackClassName);
277     return newInstance(type, fallbackClassName, null, true);
278 }
279
280 /*
281  * Try to find provider using the ServiceLoader API
282  *
283  * @param type Base class / Service interface of the factory to find.
284  *
285  * @return instance of provider class if found or null
286  */
287 private static <T> T findServiceProvider(final Class<T> type) {
288     try {
289         return AccessController.doPrivileged(new PrivilegedAction<T>() {
290             public T run() {
291                 final ServiceLoader<T> serviceLoader = ServiceLoader.load(type);
292                 final Iterator<T> iterator = serviceLoader.iterator();
293                 if (iterator.hasNext()) {
294                     return iterator.next();
295                 } else {
296                     return null;
297                 }
298             }
299         });
300     } catch (ServiceConfigurationError e) {
301         // It is not possible to wrap an error directly in
302         // FactoryConfigurationError - so we need to wrap the
303         // ServiceConfigurationError in a RuntimeException.
304         // The alternative would be to modify the logic in
305         // FactoryConfigurationError to allow setting a
306         // Throwable as the cause, but that could cause
307         // compatibility issues down the road.
308         final RuntimeException x = new RuntimeException(
309             "Provider for " + type + " cannot be created", e);
310         final FactoryConfigurationError error =
311             new FactoryConfigurationError(x, x.getMessage());
312         throw error;
313     }
314 }
315
316 }

```

## 16.5 ParserConfigurationException.java

Secuencia 158: javax/xml/parsers/ParserConfigurationException.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```

9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.parsers;
27
28 /**
29  * Indicates a serious configuration error.
30  *
31  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
32  */
33
34 public class ParserConfigurationException extends Exception {
35
36     /**
37      * Create a new <code>ParserConfigurationException</code> with no
38      * detail message.
39      */
40
41     public ParserConfigurationException() {
42         super();
43     }
44
45     /**
46      * Create a new <code>ParserConfigurationException</code> with
47      * the <code>String</code> specified as an error message.
48      *
49      * @param msg The error message for the exception.
50      */
51
52     public ParserConfigurationException(String msg) {
53         super(msg);
54     }
55
56 }

```

## 16.6 SAXParser.java

Secuencia 159: javax/xml/parsers/SAXParser.java

```

1  /*

```

```
2  * Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
```

```
25
26 package javax.xml.parsers;
```

```
27
28 import java.io.File;
```

```
29 import java.io.IOException;
```

```
30 import java.io.InputStream;
```

```
31
```

```
32 import javax.xml.validation.Schema;
```

```
33
```

```
34 import org.xml.sax.HandlerBase;
```

```
35 import org.xml.sax.InputSource;
```

```
36 import org.xml.sax.Parser;
```

```
37 import org.xml.sax.SAXException;
```

```
38 import org.xml.sax.SAXNotRecognizedException;
```

```
39 import org.xml.sax.SAXNotSupportedException;
```

```
40 import org.xml.sax.XMLReader;
```

```
41 import org.xml.sax.helpers.DefaultHandler;
```

```
42
```

```
43
```

```
44 /**
```

```
45  * Defines the API that wraps an {@link org.xml.sax.XMLReader}
46  * implementation class. In JAXP 1.0, this class wrapped the
47  * {@link org.xml.sax.Parser} interface, however this interface was
48  * replaced by the {@link org.xml.sax.XMLReader}. For ease
49  * of transition, this class continues to support the same name
50  * and interface as well as supporting new methods.
51  *
```

```
52  * An instance of this class can be obtained from the
```

```
53  * {@link javax.xml.parsers.SAXParserFactory#newSAXParser()} method.
```

```
54  * Once an instance of this class is obtained, XML can be parsed from
```

```

55  * a variety of input sources. These input sources are InputStreams,
56  * Files, URLs, and SAX InputSources.<p>
57  *
58  * This static method creates a new factory instance based
59  * on a system property setting or uses the platform default
60  * if no property has been defined.<p>
61  *
62  * The system property that controls which Factory implementation
63  * to create is named <code>"javax.xml.parsers.SAXParserFactory"</code>.
64  * This property names a class that is a concrete subclass of this
65  * abstract class. If no property is defined, a platform default
66  * will be used.</p>
67  *
68  * As the content is parsed by the underlying parser, methods of the
69  * given {@link org.xml.sax.HandlerBase} or the
70  * {@link org.xml.sax.helpers.DefaultHandler} are called.<p>
71  *
72  * Implementors of this class which wrap an underlying implementation
73  * can consider using the {@link org.xml.sax.helpers.ParserAdapter}
74  * class to initially adapt their SAX1 implementation to work under
75  * this revised class.
76  *
77  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
78  */
79  public abstract class SAXParser {
80
81      /**
82       * <p>Protected constructor to prevent instantiation.
83       * Use {@link javax.xml.parsers.SAXParserFactory#newSAXParser()}.</p>
84       */
85      protected SAXParser () {
86
87      }
88
89      /**
90       * <p>Reset this <code>SAXParser</code> to its original configuration.</p>
91       *
92       * <p><code>SAXParser</code> is reset to the same state as when it was created with
93       * {@link SAXParserFactory#newSAXParser()}.
94       * <code>reset()</code> is designed to allow the reuse of existing <code>SAXParser</code>s
95       * thus saving resources associated with the creation of new <code>SAXParser</code>s.</p>
96       *
97       * <p>The reset <code>SAXParser</code> is not guaranteed to have the same {@link Schema}
98       * <code>Object</code>, e.g. {@link Object#equals(Object obj)}. It is guaranteed to have a
99       * functionally equal
100      * <code>Schema</code>.</p>
101      *
102      * @throws UnsupportedOperationException When Implementations do not
103      * override this method
104      *
105      * @since 1.5
106      */
107      public void reset() {

```

```

107         // implementors should override this method
108         throw new UnsupportedOperationException(
109             "This SAXParser, \"" + this.getClass().getName() + "\", does not support
110             the reset functionality."
111             + " Specification \"" + this.getClass().getPackage().getSpecificationTitle
112             () + "\"
113             + " version \"" + this.getClass().getPackage().getSpecificationVersion() +
114             "\"
115             );
116     }
117
118 /**
119  * <p>Parse the content of the given {@link java.io.InputStream}
120  * instance as XML using the specified {@link org.xml.sax.HandlerBase}.
121  * <i> Use of the DefaultHandler version of this method is recommended as
122  * the HandlerBase class has been deprecated in SAX 2.0</i>.</p>
123  *
124  * @param is InputStream containing the content to be parsed.
125  * @param hb The SAX HandlerBase to use.
126  *
127  * @throws IllegalArgumentException If the given InputStream is null.
128  * @throws SAXException If parse produces a SAX error.
129  * @throws IOException If an IO error occurs interacting with the
130  *     <code>InputStream</code>.
131  *
132  * @see org.xml.sax.DocumentHandler
133  */
134 public void parse(InputStream is, HandlerBase hb)
135     throws SAXException, IOException {
136     if (is == null) {
137         throw new IllegalArgumentException("InputStream cannot be null");
138     }
139
140     InputSource input = new InputSource(is);
141     this.parse(input, hb);
142 }
143
144 /**
145  * <p>Parse the content of the given {@link java.io.InputStream}
146  * instance as XML using the specified {@link org.xml.sax.HandlerBase}.
147  * <i> Use of the DefaultHandler version of this method is recommended as
148  * the HandlerBase class has been deprecated in SAX 2.0</i>.</p>
149  *
150  * @param is InputStream containing the content to be parsed.
151  * @param hb The SAX HandlerBase to use.
152  * @param systemId The systemId which is needed for resolving relative URIs.
153  *
154  * @throws IllegalArgumentException If the given <code>InputStream</code> is
155  *     <code>null</code>.
156  * @throws IOException If any IO error occurs interacting with the
157  *     <code>InputStream</code>.
158  * @throws SAXException If any SAX errors occur during processing.

```

```

157     *
158     * @see org.xml.sax.DocumentHandler version of this method instead.
159     */
160     public void parse(
161         InputStream is,
162         HandlerBase hb,
163         String systemId)
164         throws SAXException, IOException {
165         if (is == null) {
166             throw new IllegalArgumentException("InputStream cannot be null");
167         }
168
169         InputSource input = new InputSource(is);
170         input.setSystemId(systemId);
171         this.parse(input, hb);
172     }
173
174     /**
175     * Parse the content of the given {@link java.io.InputStream}
176     * instance as XML using the specified
177     * {@link org.xml.sax.helpers.DefaultHandler}.
178     *
179     * @param is InputStream containing the content to be parsed.
180     * @param dh The SAX DefaultHandler to use.
181     *
182     * @throws IllegalArgumentException If the given InputStream is null.
183     * @throws IOException If any IO errors occur.
184     * @throws SAXException If any SAX errors occur during processing.
185     *
186     * @see org.xml.sax.DocumentHandler
187     */
188     public void parse(InputStream is, DefaultHandler dh)
189         throws SAXException, IOException {
190         if (is == null) {
191             throw new IllegalArgumentException("InputStream cannot be null");
192         }
193
194         InputSource input = new InputSource(is);
195         this.parse(input, dh);
196     }
197
198     /**
199     * Parse the content of the given {@link java.io.InputStream}
200     * instance as XML using the specified
201     * {@link org.xml.sax.helpers.DefaultHandler}.
202     *
203     * @param is InputStream containing the content to be parsed.
204     * @param dh The SAX DefaultHandler to use.
205     * @param systemId The systemId which is needed for resolving relative URIs.
206     *
207     * @throws IllegalArgumentException If the given InputStream is null.
208     * @throws IOException If any IO errors occur.
209     * @throws SAXException If any SAX errors occur during processing.

```

```

210     *
211     * @see org.xml.sax.DocumentHandler version of this method instead.
212     */
213     public void parse(
214         InputStream is,
215         DefaultHandler dh,
216         String systemId)
217         throws SAXException, IOException {
218         if (is == null) {
219             throw new IllegalArgumentException("InputStream cannot be null");
220         }
221
222         InputSource input = new InputSource(is);
223         input.setSystemId(systemId);
224         this.parse(input, dh);
225     }
226
227     /**
228     * Parse the content described by the giving Uniform Resource
229     * Identifier (URI) as XML using the specified
230     * {@link org.xml.sax.HandlerBase}.
231     * <i> Use of the DefaultHandler version of this method is recommended as
232     * the <code>HandlerBase</code> class has been deprecated in SAX 2.0</i>
233     *
234     * @param uri The location of the content to be parsed.
235     * @param hb The SAX HandlerBase to use.
236     *
237     * @throws IllegalArgumentException If the uri is null.
238     * @throws IOException If any IO errors occur.
239     * @throws SAXException If any SAX errors occur during processing.
240     *
241     * @see org.xml.sax.DocumentHandler
242     */
243     public void parse(String uri, HandlerBase hb)
244         throws SAXException, IOException {
245         if (uri == null) {
246             throw new IllegalArgumentException("uri cannot be null");
247         }
248
249         InputSource input = new InputSource(uri);
250         this.parse(input, hb);
251     }
252
253     /**
254     * Parse the content described by the giving Uniform Resource
255     * Identifier (URI) as XML using the specified
256     * {@link org.xml.sax.helpers.DefaultHandler}.
257     *
258     * @param uri The location of the content to be parsed.
259     * @param dh The SAX DefaultHandler to use.
260     *
261     * @throws IllegalArgumentException If the uri is null.
262     * @throws IOException If any IO errors occur.

```

```

263     * @throws SAXException If any SAX errors occur during processing.
264     *
265     * @see org.xml.sax.DocumentHandler
266     */
267     public void parse(String uri, DefaultHandler dh)
268         throws SAXException, IOException {
269         if (uri == null) {
270             throw new IllegalArgumentException("uri cannot be null");
271         }
272
273         InputSource input = new InputSource(uri);
274         this.parse(input, dh);
275     }
276
277     /**
278     * Parse the content of the file specified as XML using the
279     * specified {@link org.xml.sax.HandlerBase}.
280     * <i> Use of the DefaultHandler version of this method is recommended as
281     * the HandlerBase class has been deprecated in SAX 2.0</i>
282     *
283     * @param f The file containing the XML to parse
284     * @param hb The SAX HandlerBase to use.
285     *
286     * @throws IllegalArgumentException If the File object is null.
287     * @throws IOException If any IO errors occur.
288     * @throws SAXException If any SAX errors occur during processing.
289     *
290     * @see org.xml.sax.DocumentHandler
291     */
292     public void parse(File f, HandlerBase hb)
293         throws SAXException, IOException {
294         if (f == null) {
295             throw new IllegalArgumentException("File cannot be null");
296         }
297
298         //convert file to appropriate URI, f.toURI().toASCIIString()
299         //converts the URI to string as per rule specified in
300         //RFC 2396,
301         InputSource input = new InputSource(f.toURI().toASCIIString());
302         this.parse(input, hb);
303     }
304
305     /**
306     * Parse the content of the file specified as XML using the
307     * specified {@link org.xml.sax.helpers.DefaultHandler}.
308     *
309     * @param f The file containing the XML to parse
310     * @param dh The SAX DefaultHandler to use.
311     *
312     * @throws IllegalArgumentException If the File object is null.
313     * @throws IOException If any IO errors occur.
314     * @throws SAXException If any SAX errors occur during processing.
315     *

```



```

316     * @see org.xml.sax.DocumentHandler
317     */
318     public void parse(File f, DefaultHandler dh)
319         throws SAXException, IOException {
320         if (f == null) {
321             throw new IllegalArgumentException("File cannot be null");
322         }
323
324         //convert file to appropriate URI, f.toURI().toASCIIString()
325         //converts the URI to string as per rule specified in
326         //RFC 2396,
327         InputSource input = new InputSource(f.toURI().toASCIIString());
328         this.parse(input, dh);
329     }
330
331     /**
332     * Parse the content given {@link org.xml.sax.InputSource}
333     * as XML using the specified
334     * {@link org.xml.sax.HandlerBase}.
335     * <i> Use of the DefaultHandler version of this method is recommended as
336     * the HandlerBase class has been deprecated in SAX 2.0</i>
337     *
338     * @param is The InputSource containing the content to be parsed.
339     * @param hb The SAX HandlerBase to use.
340     *
341     * @throws IllegalArgumentException If the <code>InputSource</code> object
342     *     is <code>>null</code>.
343     * @throws IOException If any IO errors occur.
344     * @throws SAXException If any SAX errors occur during processing.
345     *
346     * @see org.xml.sax.DocumentHandler
347     */
348     public void parse(InputSource is, HandlerBase hb)
349         throws SAXException, IOException {
350         if (is == null) {
351             throw new IllegalArgumentException("InputSource cannot be null");
352         }
353
354         Parser parser = this.getParser();
355         if (hb != null) {
356             parser.setDocumentHandler(hb);
357             parser.setEntityResolver(hb);
358             parser.setErrorHandler(hb);
359             parser.setDTDHandler(hb);
360         }
361         parser.parse(is);
362     }
363
364     /**
365     * Parse the content given {@link org.xml.sax.InputSource}
366     * as XML using the specified
367     * {@link org.xml.sax.helpers.DefaultHandler}.
368     *

```

```

369     * @param is The InputSource containing the content to be parsed.
370     * @param dh The SAX DefaultHandler to use.
371     *
372     * @throws IllegalArgumentException If the <code>InputSource</code> object
373     *     is <code>null</code>.
374     * @throws IOException If any IO errors occur.
375     * @throws SAXException If any SAX errors occur during processing.
376     *
377     * @see org.xml.sax.DocumentHandler
378     */
379     public void parse(InputSource is, DefaultHandler dh)
380         throws SAXException, IOException {
381         if (is == null) {
382             throw new IllegalArgumentException("InputSource cannot be null");
383         }
384
385         XMLReader reader = this.getXMLReader();
386         if (dh != null) {
387             reader.setContentHandler(dh);
388             reader.setEntityResolver(dh);
389             reader.setErrorHandler(dh);
390             reader.setDTDHandler(dh);
391         }
392         reader.parse(is);
393     }
394
395     /**
396     * Returns the SAX parser that is encapsulated by the
397     * implementation of this class.
398     *
399     * @return The SAX parser that is encapsulated by the
400     *     implementation of this class.
401     *
402     * @throws SAXException If any SAX errors occur during processing.
403     */
404     public abstract org.xml.sax.Parser getParser() throws SAXException;
405
406     /**
407     * Returns the {@link org.xml.sax.XMLReader} that is encapsulated by the
408     * implementation of this class.
409     *
410     * @return The XMLReader that is encapsulated by the
411     *     implementation of this class.
412     *
413     * @throws SAXException If any SAX errors occur during processing.
414     */
415
416     public abstract org.xml.sax.XMLReader getXMLReader() throws SAXException;
417
418     /**
419     * Indicates whether or not this parser is configured to
420     * understand namespaces.
421     *

```

```

422     * @return true if this parser is configured to
423     *         understand namespaces; false otherwise.
424     */
425
426     public abstract boolean isNamespaceAware();
427
428     /**
429     * Indicates whether or not this parser is configured to
430     * validate XML documents.
431     *
432     * @return true if this parser is configured to
433     *         validate XML documents; false otherwise.
434     */
435
436     public abstract boolean isValidating();
437
438     /**
439     * <p>Sets the particular property in the underlying implementation of
440     * {@link org.xml.sax.XMLReader}.
441     * A list of the core features and properties can be found at
442     * <a href="http://sax.sourceforge.net/?selected=get-set">
443     * http://sax.sourceforge.net/?selected=get-set</a>.</p>
444     * <p>
445     * All implementations that implement JAXP 1.5 or newer are required to
446     * support the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} and
447     * {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} properties.
448     * </p>
449     * <ul>
450     * <li>
451     * <p>
452     * Setting the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} property
453     * restricts the access to external DTDs, external Entity References to
454     * the protocols specified by the property. If access is denied during parsing
455     * due to the restriction of this property, {@link org.xml.sax.SAXException}
456     * will be thrown by the parse methods defined by {@link javax.xml.parsers.SAXParser}.
457     * </p>
458     * <p>
459     * Setting the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} property
460     * restricts the access to external Schema set by the schemaLocation attribute to
461     * the protocols specified by the property. If access is denied during parsing
462     * due to the restriction of this property, {@link org.xml.sax.SAXException}
463     * will be thrown by the parse methods defined by the {@link javax.xml.parsers.SAXParser}.
464     * </p>
465     * </li>
466     * </ul>
467     *
468     * @param name The name of the property to be set.
469     * @param value The value of the property to be set.
470     *
471     * @throws SAXNotRecognizedException When the underlying XMLReader does
472     *         not recognize the property name.
473     * @throws SAXNotSupportedException When the underlying XMLReader
474     *         recognizes the property name but doesn't support the property.

```

```

475     *
476     * @see org.xml.sax.XMLReader#setProperty
477     */
478     public abstract void setProperty(String name, Object value)
479         throws SAXNotRecognizedException, SAXNotSupportedException;
480
481     /**
482     * <p>Returns the particular property requested for in the underlying
483     * implementation of {@link org.xml.sax.XMLReader}.</p>
484     *
485     * @param name The name of the property to be retrieved.
486     * @return Value of the requested property.
487     *
488     * @throws SAXNotRecognizedException When the underlying XMLReader does
489     *     not recognize the property name.
490     * @throws SAXNotSupportedException When the underlying XMLReader
491     *     recognizes the property name but doesn't support the property.
492     *
493     * @see org.xml.sax.XMLReader#getProperty
494     */
495     public abstract Object getProperty(String name)
496         throws SAXNotRecognizedException, SAXNotSupportedException;
497
498     /** <p>Get current state of canonicalization.</p>
499     *
500     * @return current state canonicalization control
501     */
502     /*
503     public boolean getCanonicalization() {
504         return canonicalState;
505     }
506     */
507
508     /** <p>Get a reference to the the {@link Schema} being used by
509     * the XML processor.</p>
510     *
511     * <p>If no schema is being used, <code>null</code> is returned.</p>
512     *
513     * @return {@link Schema} being used or <code>null</code>
514     *     if none in use
515     *
516     * @throws UnsupportedOperationException When implementation does not
517     *     override this method
518     *
519     * @since 1.5
520     */
521     public Schema getSchema() {
522         throw new UnsupportedOperationException(
523             "This parser does not support specification \""
524             + this.getClass().getPackage().getSpecificationTitle()
525             + "\" version \""
526             + this.getClass().getPackage().getSpecificationVersion()
527             + "\"")

```

```

528         );
529     }
530
531     /**
532      * <p>Get the XInclude processing mode for this parser.</p>
533      *
534      * @return
535      *     the return value of
536      *     the {@link SAXParserFactory#isXIncludeAware()}
537      *     when this parser was created from factory.
538      *
539      * @throws UnsupportedOperationException When implementation does not
540      *     override this method
541      *
542      * @since 1.5
543      *
544      * @see SAXParserFactory#setXIncludeAware(boolean)
545      */
546     public boolean isXIncludeAware() {
547         throw new UnsupportedOperationException(
548             "This parser does not support specification \""
549             + this.getClass().getPackage().getSpecificationTitle()
550             + "\" version \""
551             + this.getClass().getPackage().getSpecificationVersion()
552             + "\""
553         );
554     }
555 }

```

## 16.7 SAXParserFactory.java

Secuencia 160: javax/xml/parsers/SAXParserFactory.java

```

1  /*
2  * Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```
22  *
23  *
24  */
25
26 package javax.xml.parsers;
27
28 import javax.xml.validation.Schema;
29 import org.xml.sax.SAXException;
30 import org.xml.sax.SAXNotRecognizedException;
31 import org.xml.sax.SAXNotSupportedException;
32
33 /**
34  * Defines a factory API that enables applications to configure and
35  * obtain a SAX based parser to parse XML documents.
36  *
37  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
38  * @author <a href="mailto:Neeraj.Bajaj@sun.com">Neeraj Bajaj</a>
39  *
40  * @version $Revision: 1.9 $, $Date: 2010/05/25 16:19:44 $
41  *
42  */
43 public abstract class SAXParserFactory {
44
45     /**
46      * <p>Should Parsers be validating?</p>
47      */
48     private boolean validating = false;
49
50     /**
51      * <p>Should Parsers be namespace aware?</p>
52      */
53     private boolean namespaceAware = false;
54
55     /**
56      * <p>Protected constructor to force use of {@link #newInstance()}.</p>
57      */
58     protected SAXParserFactory () {
59
60     }
61
62     /**
63      * Obtain a new instance of a <code>SAXParserFactory</code>. This
64      * static method creates a new factory instance
65      * This method uses the following ordered lookup procedure to determine
66      * the <code>SAXParserFactory</code> implementation class to
67      * load:
68      * <ul>
69      * <li>
70      * Use the <code>javax.xml.parsers.SAXParserFactory</code> system
71      * property.
72      * </li>
73      * <li>
74      * Use the properties file "lib/jaxp.properties" in the JRE directory.
```

```

75      * This configuration file is in standard <code>java.util.Properties
76      * </code> format and contains the fully qualified name of the
77      * implementation class with the key being the system property defined
78      * above.
79      *
80      * The jaxp.properties file is read only once by the JAXP implementation
81      * and it's values are then cached for future use. If the file does not exist
82      * when the first attempt is made to read from it, no further attempts are
83      * made to check for its existence. It is not possible to change the value
84      * of any property in jaxp.properties after it has been read for the first time.
85      * </li>
86      * <li>
87      * Use the service-provider loading facilities, defined by the
88      * {@link java.util.ServiceLoader} class, to attempt to locate and load an
89      * implementation of the service using the {@linkplain
90      * java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
91      * the service-provider loading facility will use the {@linkplain
92      * java.lang.Thread#getContextClassLoader() current thread's context class loader}
93      * to attempt to load the service. If the context class
94      * loader is null, the {@linkplain
95      * ClassLoader#getClassLoader() system class loader} will be used.
96      * </li>
97      * <li>
98      * Otherwise the system-default implementation is returned.
99      * </li>
100     * </ul>
101     *
102     * Once an application has obtained a reference to a
103     * <code>SAXParserFactory</code> it can use the factory to
104     * configure and obtain parser instances.
105     *
106     *
107     *
108     * <h2>Tip for Trouble-shooting</h2>
109     * <p>Setting the <code>jaxp.debug</code> system property will cause
110     * this method to print a lot of debug messages
111     * to <code>System.err</code> about what it is doing and where it is looking at.</p>
112     *
113     * <p> If you have problems loading {@link SAXParser}s, try:</p>
114     * <pre>
115     * java -Djaxp.debug=1 YourProgram ....
116     * </pre>
117     *
118     *
119     * @return A new instance of a SAXParserFactory.
120     *
121     * @throws FactoryConfigurationError in case of {@linkplain
122     * java.util.ServiceConfigurationError service configuration error} or if
123     * the implementation is not available or cannot be instantiated.
124     */
125
126     public static SAXParserFactory newInstance() {
127         return FactoryFinder.find(

```

```

128         /* The default property name according to the JAXP spec */
129         SAXParserFactory.class,
130         /* The fallback implementation class name */
131         "com.sun.org.apache.xerces.internal.jaxp.SAXParserFactoryImpl");
132     }
133
134     /**
135      * <p>Obtain a new instance of a <code>SAXParserFactory</code> from class name.
136      * This function is useful when there are multiple providers in the classpath.
137      * It gives more control to the application as it can specify which provider
138      * should be loaded.</p>
139      *
140      * <p>Once an application has obtained a reference to a <code>SAXParserFactory</code>
141      * it can use the factory to configure and obtain parser instances.</p>
142      *
143      *
144      * <h2>Tip for Trouble-shooting</h2>
145      * <p>Setting the <code>jaxp.debug</code> system property will cause
146      * this method to print a lot of debug messages
147      * to <code>System.err</code> about what it is doing and where it is looking at.</p>
148      *
149      * <p>If you have problems, try:</p>
150      * <pre>
151      * java -Djaxp.debug=1 YourProgram ....
152      * </pre>
153      *
154      * @param factoryClassName fully qualified factory class name that provides implementation of <
155      *     code>javax.xml.parsers.SAXParserFactory</code>.
156      *
157      * @param classLoader <code>ClassLoader</code> used to load the factory class. If <code>null</
158      *     code>
159      *     current <code>Thread</code>'s context classLoader is used to load the
160      *     factory class.
161      *
162      * @return New instance of a <code>SAXParserFactory</code>
163      *
164      * @throws FactoryConfigurationError if <code>factoryClassName</code> is <code>null</code>, or
165      *     the factory class cannot be loaded, instantiated.
166      *
167      * @see #newInstance()
168      *
169      * @since 1.6
170      */
171     public static SAXParserFactory newInstance(String factoryClassName, ClassLoader classLoader){
172         //do not fallback if given classloader can't find the class, throw exception
173         return FactoryFinder.newInstance(SAXParserFactory.class,
174             factoryClassName, classLoader, false);
175     }
176
177     /**
178      * <p>Creates a new instance of a SAXParser using the currently
179      * configured factory parameters.</p>
180      *
181      *

```



```
178     * @return A new instance of a SAXParser.
179     *
180     * @throws ParserConfigurationException if a parser cannot
181     *     be created which satisfies the requested configuration.
182     * @throws SAXException for SAX errors.
183     */
184
185     public abstract SAXParser newSAXParser()
186         throws ParserConfigurationException, SAXException;
187
188
189     /**
190     * Specifies that the parser produced by this code will
191     * provide support for XML namespaces. By default the value of this is set
192     * to false.
193     *
194     * @param awareness true if the parser produced by this code will
195     *     provide support for XML namespaces; false otherwise.
196     */
197
198     public void setNamespaceAware(boolean awareness) {
199         this.namespaceAware = awareness;
200     }
201
202     /**
203     * Specifies that the parser produced by this code will
204     * validate documents as they are parsed. By default the value of this is
205     * set to false.
206     *
207     * <p>
208     * Note that "the validation" here means
209     * <a href="http://www.w3.org/TR/REC-xml#proc-types">a validating
210     * parser</a> as defined in the XML recommendation.
211     * In other words, it essentially just controls the DTD validation.
212     * (except the legacy two properties defined in JAXP 1.2.)
213     * </p>
214     *
215     * <p>
216     * To use modern schema languages such as W3C XML Schema or
217     * RELAX NG instead of DTD, you can configure your parser to be
218     * a non-validating parser by leaving the {@link #setValidating(boolean)}
219     * method false, then use the {@link #setSchema(Schema)}
220     * method to associate a schema to a parser.
221     * </p>
222     *
223     * @param validating true if the parser produced by this code will
224     *     validate documents as they are parsed; false otherwise.
225     */
226
227     public void setValidating(boolean validating) {
228         this.validating = validating;
229     }
230
```

```
231 /**
232  * Indicates whether or not the factory is configured to produce
233  * parsers which are namespace aware.
234  *
235  * @return true if the factory is configured to produce
236  *         parsers which are namespace aware; false otherwise.
237  */
238
239 public boolean isNamespaceAware() {
240     return namespaceAware;
241 }
242
243 /**
244  * Indicates whether or not the factory is configured to produce
245  * parsers which validate the XML content during parse.
246  *
247  * @return true if the factory is configured to produce parsers which validate
248  *         the XML content during parse; false otherwise.
249  */
250
251 public boolean isValidating() {
252     return validating;
253 }
254
255 /**
256  *
257  * <p>Sets the particular feature in the underlying implementation of
258  * org.xml.sax.XMLReader.
259  * A list of the core features and properties can be found at
260  * <a href="http://www.saxproject.org/">http://www.saxproject.org</a></p>
261  *
262  * <p>All implementations are required to support the {@link javax.xml.XMLConstants#
263  *     FEATURE_SECURE_PROCESSING} feature.
264  * When the feature is</p>
265  * <ul>
266  * <li>
267  *     <code>true</code>: the implementation will limit XML processing to conform to
268  *     implementation limits.
269  *     Examples include entity expansion limits and XML Schema constructs that would consume
270  *     large amounts of resources.
271  *     If XML processing is limited for security reasons, it will be reported via a call to the
272  *     registered
273  *     {@link org.xml.sax.ErrorHandler#fatalError(SAXParseException exception)}.
274  *     See {@link SAXParser} <code>parse</code> methods for handler specification.
275  * </li>
276  * <li>
277  *     When the feature is <code>false</code>, the implementation will processing XML according
278  *     to the XML specifications without
279  *     regard to possible implementation limits.
280  * </li>
281  * </ul>
282  *
283  * @param name The name of the feature to be set.
```

```

279     * @param value The value of the feature to be set.
280     *
281     * @throws ParserConfigurationException if a parser cannot
282     *     be created which satisfies the requested configuration.
283     * @throws SAXNotRecognizedException When the underlying XMLReader does
284     *     not recognize the property name.
285     * @throws SAXNotSupportedException When the underlying XMLReader
286     *     recognizes the property name but doesn't support the
287     *     property.
288     * @throws NullPointerException If the <code>name</code> parameter is null.
289     *
290     * @see org.xml.sax.XMLReader#setFeature
291     */
292     public abstract void setFeature(String name, boolean value)
293         throws ParserConfigurationException, SAXNotRecognizedException,
294             SAXNotSupportedException;
295
296     /**
297     *
298     * <p>Returns the particular property requested for in the underlying
299     * implementation of org.xml.sax.XMLReader.</p>
300     *
301     * @param name The name of the property to be retrieved.
302     *
303     * @return Value of the requested property.
304     *
305     * @throws ParserConfigurationException if a parser cannot be created which satisfies the
306     *     requested configuration.
307     * @throws SAXNotRecognizedException When the underlying XMLReader does not recognize the
308     *     property name.
309     * @throws SAXNotSupportedException When the underlying XMLReader recognizes the property name
310     *     but doesn't support the property.
311     *
312     * @see org.xml.sax.XMLReader#getProperty
313     */
314     public abstract boolean getFeature(String name)
315         throws ParserConfigurationException, SAXNotRecognizedException,
316             SAXNotSupportedException;
317
318     /**
319     * Gets the {@link Schema} object specified through
320     * the {@link #setSchema(Schema schema)} method.
321     *
322     * @throws UnsupportedOperationException When implementation does not
323     *     override this method
324     *
325     * @return
326     *     the {@link Schema} object that was last set through
327     *     the {@link #setSchema(Schema)} method, or null
328     *     if the method was not invoked since a {@link SAXParserFactory}
329     *     is created.

```

```

329     *
330     * @since 1.5
331     */
332     public Schema getSchema() {
333         throw new UnsupportedOperationException(
334             "This parser does not support specification \""
335             + this.getClass().getPackage().getSpecificationTitle()
336             + "\" version \""
337             + this.getClass().getPackage().getSpecificationVersion()
338             + "\""
339         );
340     }
341
342     /**
343     * <p>Set the {@link Schema} to be used by parsers created
344     * from this factory.</p>
345     *
346     * <p>When a {@link Schema} is non-null, a parser will use a validator
347     * created from it to validate documents before it passes information
348     * down to the application.</p>
349     *
350     * <p>When warnings/errors/fatal errors are found by the validator, the parser must
351     * handle them as if those errors were found by the parser itself.
352     * In other words, if the user-specified {@link org.xml.sax.ErrorHandler}
353     * is set, it must receive those errors, and if not, they must be
354     * treated according to the implementation specific
355     * default error handling rules.
356     *
357     * <p>A validator may modify the SAX event stream (for example by
358     * adding default values that were missing in documents), and a parser
359     * is responsible to make sure that the application will receive
360     * those modified event stream.</p>
361     *
362     * <p>Initially, null is set as the {@link Schema}.</p>
363     *
364     * <p>This processing will take effect even if
365     * the {@link #isValidating()} method returns false.
366     *
367     * <p>It is an error to use
368     * the http://java.sun.com/xml/jaxp/properties/schemaSource
369     * property and/or the http://java.sun.com/xml/jaxp/properties/schemaLanguage
370     * property in conjunction with a non-null {@link Schema} object.
371     * Such configuration will cause a {@link SAXException}
372     * exception when those properties are set on a {@link SAXParser}.</p>
373     *
374     * <h4>Note for implementors</h4>
375     * <p>
376     * A parser must be able to work with any {@link Schema}
377     * implementation. However, parsers and schemas are allowed
378     * to use implementation-specific custom mechanisms
379     * as long as they yield the result described in the specification.
380     * </p>
381     *

```

```

382     * @param schema <code>Schema</code> to use, <code>null</code> to remove a schema.
383     *
384     * @throws UnsupportedOperationException When implementation does not
385     *     override this method
386     *
387     * @since 1.5
388     */
389     public void setSchema(Schema schema) {
390         throw new UnsupportedOperationException(
391             "This parser does not support specification \""
392             + this.getClass().getPackage().getSpecificationTitle()
393             + "\" version \""
394             + this.getClass().getPackage().getSpecificationVersion()
395             + "\""
396         );
397     }
398
399     /**
400     * <p>Set state of XInclude processing.</p>
401     *
402     * <p>If XInclude markup is found in the document instance, should it be
403     * processed as specified in <a href="http://www.w3.org/TR/xinclude/">
404     * XML Inclusions (XInclude) Version 1.0</a>.</p>
405     *
406     * <p>XInclude processing defaults to <code>>false</code>.</p>
407     *
408     * @param state Set XInclude processing to <code>true</code> or
409     *     <code>>false</code>
410     *
411     * @throws UnsupportedOperationException When implementation does not
412     *     override this method
413     *
414     * @since 1.5
415     */
416     public void setXIncludeAware(final boolean state) {
417         if (state) {
418             throw new UnsupportedOperationException(" setXIncludeAware " +
419                 "is not supported on this JAXP" +
420                 " implementation or earlier: " + this.getClass());
421         }
422     }
423
424     /**
425     * <p>Get state of XInclude processing.</p>
426     *
427     * @return current state of XInclude processing
428     *
429     * @throws UnsupportedOperationException When implementation does not
430     *     override this method
431     *
432     * @since 1.5
433     */
434     public boolean isXIncludeAware() {

```

```

435         throw new UnsupportedOperationException(
436             "This parser does not support specification \""
437             + this.getClass().getPackage().getSpecificationTitle()
438             + "\" version \""
439             + this.getClass().getPackage().getSpecificationVersion()
440             + "\"\"
441             );
442     }
443 }

```

## 16.8 SecuritySupport.java

Secuencia 161: javax/xml/parsers/SecuritySupport.java

```

1  /*
2  * Copyright (c) 2004, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.parsers;
27
28 import java.security.*;
29 import java.net.*;
30 import java.io.*;
31 import java.util.*;
32
33 /**
34  * This class is duplicated for each JAXP subpackage so keep it in sync.
35  * It is package private and therefore is not exposed as part of the JAXP
36  * API.
37  *
38  * Security related methods that only work on J2SE 1.2 and newer.
39  */
40 class SecuritySupport {

```

```
41
42
43 ClassLoader getContextClassLoader() throws SecurityException{
44     return (ClassLoader)
45         AccessController.doPrivileged(new PrivilegedAction() {
46             public Object run() {
47                 ClassLoader cl = null;
48                 //try {
49                     cl = Thread.currentThread().getContextClassLoader();
50                 //} catch (SecurityException ex) { }
51
52                 if (cl == null)
53                     cl = ClassLoader.getSystemClassLoader();
54
55                 return cl;
56             }
57         });
58 }
59
60 String getSystemProperty(final String propName) {
61     return (String)
62         AccessController.doPrivileged(new PrivilegedAction() {
63             public Object run() {
64                 return System.getProperty(propName);
65             }
66         });
67 }
68
69 FileInputStream getFileInputStream(final File file)
70     throws FileNotFoundException
71 {
72     try {
73         return (FileInputStream)
74             AccessController.doPrivileged(new PrivilegedExceptionAction() {
75                 public Object run() throws FileNotFoundException {
76                     return new FileInputStream(file);
77                 }
78             });
79     } catch (PrivilegedActionException e) {
80         throw (FileNotFoundException)e.getException();
81     }
82 }
83
84 InputStream getResourceAsStream(final ClassLoader cl,
85     final String name)
86 {
87     return (InputStream)
88         AccessController.doPrivileged(new PrivilegedAction() {
89             public Object run() {
90                 InputStream ris;
91                 if (cl == null) {
92                     ris = Object.class.getResourceAsStream(name);
93                 } else {
```

```

94         ris = cl.getResourceAsStream(name);
95     }
96     return ris;
97 }
98 });
99 }
100
101 boolean doesFileExist(final File f) {
102     return ((Boolean)
103         AccessController.doPrivileged(new PrivilegedAction() {
104             public Object run() {
105                 return new Boolean(f.exists());
106             }
107         })).booleanValue();
108 }
109
110 }

```

## 17 Xml Soap (javax.xml.soap package)

### 17.1 AttachmentPart.java

Secuencia 162: javax/xml/soap/AttachmentPart.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import java.io.InputStream;
29 import java.io.Reader;
30 import java.util.Iterator;

```



```

31
32 import javax.activation.DataHandler;
33
34 /**
35  * A single attachment to a SOAPMessage object. A SOAPMessage
36  * object may contain zero, one, or many AttachmentPart objects.
37  * Each AttachmentPart object consists of two parts,
38  * application-specific content and associated MIME headers. The
39  * MIME headers consists of name/value pairs that can be used to
40  * identify and describe the content.
41  * <p>
42  * An AttachmentPart object must conform to certain standards.
43  * <OL>
44  * <LI>It must conform to <a href="http://www.ietf.org/rfc/rfc2045.txt">
45  *   MIME [RFC2045] standards</a>
46  * <LI>It MUST contain content
47  * <LI>The header portion MUST include the following header:
48  *   <UL>
49  *     <LI><code>Content-Type</code><br>
50  *       This header identifies the type of data in the content of an
51  *       AttachmentPart object and MUST conform to [RFC2045].
52  *       The following is an example of a Content-Type header:
53  *       <PRE>
54  *       Content-Type: application/xml
55  *       </PRE>
56  *       The following line of code, in which ap is an
57  *       AttachmentPart object, sets the header shown in
58  *       the previous example.
59  *       <PRE>
60  *       ap.setMimeHeader("Content-Type", "application/xml");
61  *       </PRE>
62  *   </UL>
63  * </OL>
64  * <p>
65  * There are no restrictions on the content portion of an 
66  * AttachmentPart object. The content may be anything from a
67  * simple plain text object to a complex XML document or image file.
68  *
69  * <p>
70  * An AttachmentPart object is created with the method
71  * SOAPMessage.createAttachmentPart. After setting its MIME headers,
72  * the AttachmentPart object is added to the message
73  * that created it with the method SOAPMessage.addAttachmentPart.
74  *
75  * <p>
76  * The following code fragment, in which m is a
77  * SOAPMessage object and contentString1 is a
78  * String, creates an instance of AttachmentPart,
79  * sets the AttachmentPart object with some content and
80  * header information, and adds the AttachmentPart object to
81  * the SOAPMessage object.
82  * <PRE>
83

```

```

84 *      AttachmentPart ap1 = m.createAttachmentPart();
85 *      ap1.setContent(contentString1, "text/plain");
86 *      m.addAttachmentPart(ap1);
87 * </PRE>
88 *
89 *
90 * <p>
91 * The following code fragment creates and adds a second
92 * <code>AttachmentPart</code> instance to the same message. <code>jpegData</code>
93 * is a binary byte buffer representing the jpeg file.
94 * <PRE>
95 *      AttachmentPart ap2 = m.createAttachmentPart();
96 *      byte[] jpegData = ...;
97 *      ap2.setContent(new ByteArrayInputStream(jpegData), "image/jpeg");
98 *      m.addAttachmentPart(ap2);
99 * </PRE>
100 * <p>
101 * The <code>getContent</code> method retrieves the contents and header from
102 * an <code>AttachmentPart</code> object. Depending on the
103 * <code>DataContentHandler</code> objects present, the returned
104 * <code>Object</code> can either be a typed Java object corresponding
105 * to the MIME type or an <code>InputStream</code> object that contains the
106 * content as bytes.
107 * <PRE>
108 *      String content1 = ap1.getContent();
109 *      java.io.InputStream content2 = ap2.getContent();
110 * </PRE>
111 *
112 * The method <code>clearContent</code> removes all the content from an
113 * <code>AttachmentPart</code> object but does not affect its header information.
114 * <PRE>
115 *      ap1.clearContent();
116 * </PRE>
117 */
118
119 public abstract class AttachmentPart {
120     /**
121      * Returns the number of bytes in this <code>AttachmentPart</code>
122      * object.
123      *
124      * @return the size of this <code>AttachmentPart</code> object in bytes
125      *         or -1 if the size cannot be determined
126      * @exception SOAPException if the content of this attachment is
127      *         corrupted or if there was an exception while trying
128      *         to determine the size.
129      */
130     public abstract int getSize() throws SOAPException;
131
132     /**
133      * Clears out the content of this <code>AttachmentPart</code> object.
134      * The MIME header portion is left untouched.
135      */
136     public abstract void clearContent();

```

```

137
138 /**
139  * Gets the content of this AttachmentPart object as a Java
140  * object. The type of the returned Java object depends on (1) the
141  * DataContentHandler object that is used to interpret the bytes
142  * and (2) the Content-Type given in the header.
143  * <p>
144  * For the MIME content types "text/plain", "text/html" and "text/xml", the
145  * DataContentHandler object does the conversions to and
146  * from the Java types corresponding to the MIME types.
147  * For other MIME types, the DataContentHandler object
148  * can return an InputStream object that contains the content data
149  * as raw bytes.
150  * <p>
151  * A SAAJ-compliant implementation must, as a minimum, return a
152  * java.lang.String object corresponding to any content
153  * stream with a Content-Type value of
154  * text/plain, a
155  * javax.xml.transform.stream.StreamSource object corresponding to a
156  * content stream with a Content-Type value of
157  * text/xml, a java.awt.Image object
158  * corresponding to a content stream with a
159  * Content-Type value of image/gif or
160  * image/jpeg. For those content types that an
161  * installed DataContentHandler object does not understand, the
162  * DataContentHandler object is required to return a
163  * java.io.InputStream object with the raw bytes.
164  *
165  * @return a Java object with the content of this AttachmentPart
166  *         object
167  *
168  * @exception SOAPException if there is no content set into this
169  *         AttachmentPart object or if there was a data
170  *         transformation error
171  */
172 public abstract Object getContent() throws SOAPException;
173
174 /**
175  * Gets the content of this AttachmentPart object as an
176  * InputStream as if a call had been made to getContent and no
177  * DataContentHandler had been registered for the
178  * content-type of this AttachmentPart.
179  * <p>
180  * Note that reading from the returned InputStream would result in consuming
181  * the data in the stream. It is the responsibility of the caller to reset
182  * the InputStream appropriately before calling a Subsequent API. If a copy
183  * of the raw attachment content is required then the {@link #getRawContentBytes} API
184  * should be used instead.
185  *
186  * @return an InputStream from which the raw data contained by
187  *         the AttachmentPart can be accessed.
188  *
189  * @throws SOAPException if there is no content set into this

```

```

190     *      <code>AttachmentPart</code> object or if there was a data
191     *      transformation error.
192     *
193     * @since SAAJ 1.3
194     * @see #getRawContentBytes
195     */
196     public abstract InputStream getRawContent() throws SOAPException;
197
198     /**
199     * Gets the content of this <code>AttachmentPart</code> object as a
200     * byte[] array as if a call had been made to <code>getContent</code> and no
201     * <code>DataContentHandler</code> had been registered for the
202     * <code>content-type</code> of this <code>AttachmentPart</code>.
203     *
204     * @return a <code>byte[]</code> array containing the raw data of the
205     *         <code>AttachmentPart</code>.
206     *
207     * @throws SOAPException if there is no content set into this
208     *         <code>AttachmentPart</code> object or if there was a data
209     *         transformation error.
210     *
211     * @since SAAJ 1.3
212     */
213     public abstract byte[] getRawContentBytes() throws SOAPException;
214
215     /**
216     * Returns an <code>InputStream</code> which can be used to obtain the
217     * content of <code>AttachmentPart</code> as Base64 encoded
218     * character data, this method would base64 encode the raw bytes
219     * of the attachment and return.
220     *
221     * @return an <code>InputStream</code> from which the Base64 encoded
222     *         <code>AttachmentPart</code> can be read.
223     *
224     * @throws SOAPException if there is no content set into this
225     *         <code>AttachmentPart</code> object or if there was a data
226     *         transformation error.
227     *
228     * @since SAAJ 1.3
229     */
230     public abstract InputStream getBase64Content() throws SOAPException;
231
232     /**
233     * Sets the content of this attachment part to that of the given
234     * <code>Object</code> and sets the value of the <code>Content-Type</code>
235     * header to the given type. The type of the
236     * <code>Object</code> should correspond to the value given for the
237     * <code>Content-Type</code>. This depends on the particular
238     * set of <code>DataContentHandler</code> objects in use.
239     *
240     *
241     * @param object the Java object that makes up the content for
242     *        this attachment part

```

```

243     * @param contentType the MIME string that specifies the type of
244     *                     the content
245     *
246     * @exception IllegalArgumentException may be thrown if the contentType
247     *                     does not match the type of the content object, or if there
248     *                     was no <code>DataContentHandler</code> object for this
249     *                     content object
250     *
251     * @see #getContent
252     */
253     public abstract void setContent(Object object, String contentType);
254
255     /**
256     * Sets the content of this attachment part to that contained by the
257     * <code>InputStream</code> <code>content</code> and sets the value of the
258     * <code>Content-Type</code> header to the value contained in
259     * <code>contentType</code>.
260     * <P>
261     * A subsequent call to getSize() may not be an exact measure
262     * of the content size.
263     *
264     * @param content the raw data to add to the attachment part
265     * @param contentType the value to set into the <code>Content-Type</code>
266     * header
267     *
268     * @exception SOAPException if an there is an error in setting the content
269     * @exception NullPointerException if <code>content</code> is null
270     * @since SAAJ 1.3
271     */
272     public abstract void setRawContent(InputStream content, String contentType) throws
        SOAPException;
273
274     /**
275     * Sets the content of this attachment part to that contained by the
276     * <code>byte[]</code> array <code>content</code> and sets the value of the
277     * <code>Content-Type</code> header to the value contained in
278     * <code>contentType</code>.
279     *
280     * @param content the raw data to add to the attachment part
281     * @param contentType the value to set into the <code>Content-Type</code>
282     * header
283     * @param offset the offset in the byte array of the content
284     * @param len the number of bytes that form the content
285     *
286     * @exception SOAPException if an there is an error in setting the content
287     * or content is null
288     * @since SAAJ 1.3
289     */
290     public abstract void setRawContentBytes(
291         byte[] content, int offset, int len, String contentType)
292         throws SOAPException;
293
294

```

```

295 /**
296  * Sets the content of this attachment part from the Base64 source
297  * <code>InputStream</code> and sets the value of the
298  * <code>Content-Type</code> header to the value contained in
299  * <code>contentType</code>, This method would first decode the base64
300  * input and write the resulting raw bytes to the attachment.
301  * <P>
302  * A subsequent call to getSize() may not be an exact measure
303  * of the content size.
304  *
305  * @param content the base64 encoded data to add to the attachment part
306  * @param contentType the value to set into the <code>Content-Type</code>
307  * header
308  *
309  * @exception SOAPException if an there is an error in setting the content
310  * @exception NullPointerException if <code>content</code> is null
311  *
312  * @since SAAJ 1.3
313  */
314 public abstract void setBase64Content(
315     InputStream content, String contentType) throws SOAPException;
316
317
318 /**
319  * Gets the <code>DataHandler</code> object for this <code>AttachmentPart</code>
320  * object.
321  *
322  * @return the <code>DataHandler</code> object associated with this
323  *         <code>AttachmentPart</code> object
324  *
325  * @exception SOAPException if there is no data in
326  * this <code>AttachmentPart</code> object
327  */
328 public abstract DataHandler getDataHandler()
329     throws SOAPException;
330
331 /**
332  * Sets the given <code>DataHandler</code> object as the data handler
333  * for this <code>AttachmentPart</code> object. Typically, on an incoming
334  * message, the data handler is automatically set. When
335  * a message is being created and populated with content, the
336  * <code>setDataHandler</code> method can be used to get data from
337  * various data sources into the message.
338  *
339  * @param dataHandler the <code>DataHandler</code> object to be set
340  *
341  * @exception IllegalArgumentException if there was a problem with
342  * the specified <code>DataHandler</code> object
343  */
344 public abstract void setDataHandler(DataHandler dataHandler);
345
346
347 /**

```

```
348     * Gets the value of the MIME header whose name is "Content-ID".
349     *
350     * @return a <code>String</code> giving the value of the
351     *         "Content-ID" header or <code>null</code> if there
352     *         is none
353     * @see #setContentId
354     */
355     public String getContentId() {
356         String[] values = getMimeHeader("Content-ID");
357         if (values != null && values.length > 0)
358             return values[0];
359         return null;
360     }
361
362     /**
363     * Gets the value of the MIME header whose name is "Content-Location".
364     *
365     * @return a <code>String</code> giving the value of the
366     *         "Content-Location" header or <code>null</code> if there
367     *         is none
368     */
369     public String getContentLocation() {
370         String[] values = getMimeHeader("Content-Location");
371         if (values != null && values.length > 0)
372             return values[0];
373         return null;
374     }
375
376     /**
377     * Gets the value of the MIME header whose name is "Content-Type".
378     *
379     * @return a <code>String</code> giving the value of the
380     *         "Content-Type" header or <code>null</code> if there
381     *         is none
382     */
383     public String getContentType() {
384         String[] values = getMimeHeader("Content-Type");
385         if (values != null && values.length > 0)
386             return values[0];
387         return null;
388     }
389
390     /**
391     * Sets the MIME header whose name is "Content-ID" with the given value.
392     *
393     * @param contentId a <code>String</code> giving the value of the
394     *         "Content-ID" header
395     *
396     * @exception IllegalArgumentException if there was a problem with
397     *         the specified <code>contentId</code> value
398     * @see #getContentId
399     */
400     public void setContentId(String contentId)
```

```
401 {
402     setMimeHeader("Content-ID", contentId);
403 }
404
405
406 /**
407  * Sets the MIME header whose name is "Content-Location" with the given value.
408  *
409  *
410  * @param contentLocation a <code>String</code> giving the value of the
411  *     "Content-Location" header
412  * @exception IllegalArgumentException if there was a problem with
413  *     the specified content location
414  */
415 public void setContentLocation(String contentLocation)
416 {
417     setMimeHeader("Content-Location", contentLocation);
418 }
419
420 /**
421  * Sets the MIME header whose name is "Content-Type" with the given value.
422  *
423  *
424  * @param contentType a <code>String</code> giving the value of the
425  *     "Content-Type" header
426  *
427  * @exception IllegalArgumentException if there was a problem with
428  *     the specified content type
429  */
430 public void setContentType(String contentType)
431 {
432     setMimeHeader("Content-Type", contentType);
433 }
434
435 /**
436  * Removes all MIME headers that match the given name.
437  *
438  *
439  * @param header the string name of the MIME header/s to
440  *     be removed
441  */
442 public abstract void removeMimeHeader(String header);
443
444 /**
445  * Removes all the MIME header entries.
446  */
447 public abstract void removeAllMimeHeaders();
448
449 /**
450  * Gets all the values of the header identified by the given
451  * <code>String</code>.
452  *
453  *
454  * @param name the name of the header; example: "Content-Type"
455  * @return a <code>String</code> array giving the value for the
```



```

454     *         specified header
455     * @see #setMimeHeader
456     */
457     public abstract String[] getMimeHeader(String name);
458
459
460     /**
461     * Changes the first header entry that matches the given name
462     * to the given value, adding a new header if no existing header
463     * matches. This method also removes all matching headers but the first. <p>
464     *
465     * Note that RFC822 headers can only contain US-ASCII characters.
466     *
467     * @param   name       a <code>String</code> giving the name of the header
468     *                   for which to search
469     * @param   value      a <code>String</code> giving the value to be set for
470     *                   the header whose name matches the given name
471     *
472     * @exception IllegalArgumentException if there was a problem with
473     *         the specified mime header name or value
474     */
475     public abstract void setMimeHeader(String name, String value);
476
477
478     /**
479     * Adds a MIME header with the specified name and value to this
480     * <code>AttachmentPart</code> object.
481     * <p>
482     * Note that RFC822 headers can contain only US-ASCII characters.
483     *
484     * @param   name       a <code>String</code> giving the name of the header
485     *                   to be added
486     * @param   value      a <code>String</code> giving the value of the header
487     *                   to be added
488     *
489     * @exception IllegalArgumentException if there was a problem with
490     *         the specified mime header name or value
491     */
492     public abstract void addMimeHeader(String name, String value);
493
494
495     /**
496     * Retrieves all the headers for this <code>AttachmentPart</code> object
497     * as an iterator over the <code>MimeHeader</code> objects.
498     *
499     * @return   an <code>Iterator</code> object with all of the Mime
500     *         headers for this <code>AttachmentPart</code> object
501     */
502     public abstract Iterator getAllMimeHeaders();
503
504     /**
505     * Retrieves all <code>MimeHeader</code> objects that match a name in
506     * the given array.
507     *

```

```

507     * @param names a <code>String</code> array with the name(s) of the
508     *           MIME headers to be returned
509     * @return  all of the MIME headers that match one of the names in the
510     *           given array as an <code>Iterator</code> object
511     */
512     public abstract Iterator getMatchingMimeHeaders(String[] names);
513
514     /**
515     * Retrieves all <code>MimeHeader</code> objects whose name does
516     * not match a name in the given array.
517     *
518     * @param names a <code>String</code> array with the name(s) of the
519     *           MIME headers not to be returned
520     * @return  all of the MIME headers in this <code>AttachmentPart</code> object
521     *           except those that match one of the names in the
522     *           given array. The nonmatching MIME headers are returned as an
523     *           <code>Iterator</code> object.
524     */
525     public abstract Iterator getNonMatchingMimeHeaders(String[] names);
526 }

```

## 17.2 Detail.java

Secuencia 163: javax/xml/soap/Detail.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import java.util.Iterator;
29

```

```

30 import javax.xml.namespace.QName;
31
32 /**
33  * A container for DetailEntry objects. DetailEntry
34  * objects give detailed error information that is application-specific and
35  * related to the SOAPBody object that contains it.
36  * <P>
37  * A Detail object, which is part of a SOAPFault
38  * object, can be retrieved using the method SOAPFault.getDetail.
39  * The Detail interface provides two methods. One creates a new
40  * DetailEntry object and also automatically adds it to
41  * the Detail object. The second method gets a list of the
42  * DetailEntry objects contained in a Detail
43  * object.
44  * <P>
45  * The following code fragment, in which sf is a SOAPFault
46  * object, gets its Detail object (d), adds a new
47  * DetailEntry object to d, and then gets a list of all the
48  * DetailEntry objects in d. The code also creates a
49  * Name object to pass to the method addDetailEntry.
50  * The variable se, used to create the Name object,
51  * is a SOAPEnvelope object.
52  * <PRE>
53  *     Detail d = sf.getDetail();
54  *     Name name = se.createName("GetLastTradePrice", "WOMBAT",
55  *                               "http://www.wombat.org/trader");
56  *     d.addDetailEntry(name);
57  *     Iterator it = d.getDetailEntries();
58  * </PRE>
59  */
60 public interface Detail extends SOAPFaultElement {
61
62     /**
63      * Creates a new DetailEntry object with the given
64      * name and adds it to this Detail object.
65      *
66      * @param name a Name object identifying the
67      *             new DetailEntry object
68      *
69      * @exception SOAPException thrown when there is a problem in adding a
70      *         DetailEntry object to this Detail object.
71      *
72      * @see Detail#addDetailEntry(QName qname)
73      */
74     public DetailEntry addDetailEntry(Name name) throws SOAPException;
75
76     /**
77      * Creates a new DetailEntry object with the given
78      * QName and adds it to this Detail object. This method
79      * is the preferred over the one using Name.
80      *
81      * @param qname a QName object identifying the
82      *              new DetailEntry object

```

```

83      *
84      * @exception SOAPException thrown when there is a problem in adding a
85      * DetailEntry object to this Detail object.
86      *
87      * @see Detail#addDetailEntry(Name name)
88      * @since SAAJ 1.3
89      */
90      public DetailEntry addDetailEntry(QName qname) throws SOAPException;
91
92      /**
93       * Gets an Iterator over all of the <code>DetailEntry</code>s in this <code>Detail</code>
94       * object.
95       *
96       * @return an <code>Iterator</code> object over the <code>DetailEntry</code>
97       *         objects in this <code>Detail</code> object
98       */
99      public Iterator getDetailEntries();

```

### 17.3 DetailEntry.java

Secuencia 164: javax/xml/soap/DetailEntry.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 /**
29  * The content for a <code>Detail</code> object, giving details for
30  * a <code>SOAPFault</code> object. A <code>DetailEntry</code> object,
31  * which carries information about errors related to the <code>SOAPBody</code>

```

```
32 * object that contains it, is application-specific.
33 */
34 public interface DetailEntry extends SOAPElement {
35
36 }
```

## 17.4 FactoryFinder.java

Secuencia 165: javax/xml/soap/FactoryFinder.java

```
1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import java.io.*;
29 import java.util.Properties;
30
31
32 class FactoryFinder {
33
34     /**
35      * Creates an instance of the specified class using the specified
36      * <code>ClassLoader</code> object.
37      *
38      * @exception SOAPException if the given class could not be found
39      *             or could not be instantiated
40      */
41     private static Object newInstance(String className,
42                                     ClassLoader classLoader)
43         throws SOAPException
44     {
```

```

45     try {
46         Class spiClass = safeLoadClass(className, classLoader);
47         return spiClass.newInstance();
48     } catch (ClassNotFoundException x) {
49         throw new SOAPException("Provider " + className + " not found", x);
50     } catch (Exception x) {
51         throw new SOAPException("Provider " + className + " could not be instantiated: " + x, x
52             );
53     }
54 }
55
56 /**
57  * Finds the implementation <code>Class</code> object for the given
58  * factory name, or null if that fails.
59  * <P>
60  * This method is package private so that this code can be shared.
61  *
62  * @return the <code>Class</code> object of the specified message factory;
63  *         or <code>null</code>
64  *
65  * @param factoryId      the name of the factory to find, which is
66  *                       a system property
67  * @exception SOAPException if there is a SOAP error
68  */
69 static Object find(String factoryId)
70     throws SOAPException
71 {
72     return find(factoryId, null, false);
73 }
74
75 /**
76  * Finds the implementation <code>Class</code> object for the given
77  * factory name, or if that fails, finds the <code>Class</code> object
78  * for the given fallback class name. The arguments supplied must be
79  * used in order. If using the first argument is successful, the second
80  * one will not be used.
81  * <P>
82  * This method is package private so that this code can be shared.
83  *
84  * @return the <code>Class</code> object of the specified message factory;
85  *         may be <code>null</code>
86  *
87  * @param factoryId      the name of the factory to find, which is
88  *                       a system property
89  * @param fallbackClassName the implementation class name, which is
90  *                       to be used only if nothing else
91  *                       is found; <code>null</code> to indicate that
92  *                       there is no fallback class name
93  * @exception SOAPException if there is a SOAP error
94  */
95 static Object find(String factoryId, String fallbackClassName)
96     throws SOAPException

```

```

97     {
98         return find(factoryId, fallbackClassName, true);
99     }
100
101     /**
102     * Finds the implementation <code>Class</code> object for the given
103     * factory name, or if that fails, finds the <code>Class</code> object
104     * for the given default class name, but only if <code>tryFallback</code>
105     * is <code>true</code>. The arguments supplied must be used in order
106     * If using the first argument is successful, the second one will not
107     * be used. Note the default class name may be needed even if fallback
108     * is not to be attempted, so certain error conditions can be handled.
109     * <P>
110     * This method is package private so that this code can be shared.
111     *
112     * @return the <code>Class</code> object of the specified message factory;
113     *         may not be <code>null</code>
114     *
115     * @param factoryId      the name of the factory to find, which is
116     *                        a system property
117     * @param defaultClassName the implementation class name, which is
118     *                        to be used only if nothing else
119     *                        is found; <code>null</code> to indicate
120     *                        that there is no default class name
121     * @param tryFallback     whether to try the default class as a
122     *                        fallback
123     * @exception SOAPException if there is a SOAP error
124     */
125     static Object find(String factoryId, String defaultClassName,
126                       boolean tryFallback) throws SOAPException {
127         ClassLoader classLoader;
128         try {
129             classLoader = Thread.currentThread().getContextClassLoader();
130         } catch (Exception x) {
131             throw new SOAPException(x.toString(), x);
132         }
133
134         // Use the system property first
135         try {
136             String systemProp =
137                 System.getProperty( factoryId );
138             if( systemProp!=null) {
139                 return newInstance(systemProp, classLoader);
140             }
141         } catch (SecurityException se) {
142         }
143
144         // try to read from $java.home/lib/jaxm.properties
145         try {
146             String javah=System.getProperty( "java.home" );
147             String configFile = javah + File.separator +
148                 "lib" + File.separator + "jaxm.properties";
149             File f=new File( configFile );

```

```

150         if( f.exists()) {
151             Properties props=new Properties();
152             props.load( new FileInputStream(f));
153             String factoryClassName = props.getProperty(factoryId);
154             return newInstance(factoryClassName, classLoader);
155         }
156     } catch(Exception ex ) {
157     }
158
159     String serviceId = "META-INF/services/" + factoryId;
160     // try to find services in CLASSPATH
161     try {
162         InputStream is=null;
163         if (classLoader == null) {
164             is=ClassLoader.getResourceAsStream(serviceId);
165         } else {
166             is=classLoader.getResourceAsStream(serviceId);
167         }
168
169         if( is!=null ) {
170             BufferedReader rd =
171                 new BufferedReader(new InputStreamReader(is, "UTF-8"));
172
173             String factoryClassName = rd.readLine();
174             rd.close();
175
176             if (factoryClassName != null &&
177                 ! "".equals(factoryClassName)) {
178                 return newInstance(factoryClassName, classLoader);
179             }
180         }
181     } catch( Exception ex ) {
182     }
183
184     // If not found and fallback should not be tried, return a null result.
185     if (!tryFallback)
186         return null;
187
188     // We didn't find the class through the usual means so try the default
189     // (built in) factory if specified.
190     if (defaultClassName == null) {
191         throw new SOAPException(
192             "Provider for " + factoryId + " cannot be found", null);
193     }
194     return newInstance(defaultClassName, classLoader);
195 }
196
197 /**
198  * Loads the class, provided that the calling thread has an access to the
199  * class being loaded. If this is the specified default factory class and it
200  * is restricted by package.access we get a SecurityException and can do a
201  * Class.forName() on it so it will be loaded by the bootstrap class loader.
202  */

```



```

203     private static Class safeLoadClass(String className,
204                                       ClassLoader classLoader)
205         throws ClassNotFoundException {
206         try {
207             // make sure that the current thread has an access to the package of the given name.
208             SecurityManager s = System.getSecurityManager();
209             if (s != null) {
210                 int i = className.lastIndexOf('.');
211                 if (i != -1) {
212                     s.checkPackageAccess(className.substring(0, i));
213                 }
214             }
215
216             if (classLoader == null)
217                 return Class.forName(className);
218             else
219                 return classLoader.loadClass(className);
220         } catch (SecurityException se) {
221             // (only) default implementation can be loaded
222             // using bootstrap class loader:
223             if (isDefaultImplementation(className))
224                 return Class.forName(className);
225
226             throw se;
227         }
228     }
229
230     private static boolean isDefaultImplementation(String className) {
231         return MessageFactory.DEFAULT_MESSAGE_FACTORY.equals(className) ||
232             SOAPFactory.DEFAULT_SOAP_FACTORY.equals(className) ||
233             SOAPConnectionFactory.DEFAULT_SOAP_CONNECTION_FACTORY.equals(className) ||
234             SAAJMetaFactory.DEFAULT_META_FACTORY_CLASS.equals(className);
235     }
236 }

```

## 17.5 MessageFactory.java

Secuencia 166: javax/xml/soap/MessageFactory.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.soap;
27
28
29  import java.io.IOException;
30  import java.io.InputStream;
31
32  /**
33   * A factory for creating SOAPMessage objects.
34   * <P>
35   * A SAAJ client can create a MessageFactory object
36   * using the method newInstance, as shown in the following
37   * lines of code.
38   * <PRE>
39   *     MessageFactory mf = MessageFactory.newInstance();
40   *     MessageFactory mf12 = MessageFactory.newInstance(SOAPConstants.SOAP_1_2_PROTOCOL);
41   * </PRE>
42   * <P>
43   * All MessageFactory objects, regardless of how they are
44   * created, will produce SOAPMessage objects that
45   * have the following elements by default:
46   * <UL>
47   * <LI>A SOAPPart object
48   * <LI>A SOAPEnvelope object
49   * <LI>A SOAPBody object
50   * <LI>A SOAPHeader object
51   * </UL>
52   * In some cases, specialized MessageFactory objects may be obtained that produce messages
53   * prepopulated with additional entries in the SOAPHeader object and the
54   * SOAPBody object.
55   * The content of a new SOAPMessage object depends on which of the two
56   * MessageFactory methods is used to create it.
57   * <UL>
58   * <LI><code>createMessage()InputStream object and headers from the
62   *     MimeHeaders object <BR>
63   *     This method can be used internally by a service implementation to
64   *     create a message that is a response to a request.
65   * </UL>
66   */
67  public abstract class MessageFactory {
68

```

```

69     static final String DEFAULT_MESSAGE_FACTORY
70         = "com.sun.xml.internal.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl";
71
72     static private final String MESSAGE_FACTORY_PROPERTY
73         = "javax.xml.soap.MessageFactory";
74
75     /**
76      * Creates a new MessageFactory object that is an instance
77      * of the default implementation (SOAP 1.1),
78      *
79      * This method uses the following ordered lookup procedure to determine the MessageFactory
80      * implementation class to load:
81      * <UL>
82      * <LI> Use the javax.xml.soap.MessageFactory system property.
83      * <LI> Use the properties file "lib/jaxm.properties" in the JRE directory. This configuration
84      * file is in standard
85      * java.util.Properties format and contains the fully qualified name of the implementation
86      * class with the key being the
87      * system property defined above.
88      * <LI> Use the Services API (as detailed in the JAR specification), if available, to
89      * determine the classname. The Services API
90      * will look for a classname in the file META-INF/services/javax.xml.soap.MessageFactory in
91      * jars available to the runtime.
92      * <LI> Use the SAAJMetaFactory instance to locate the MessageFactory implementation class.
93      * </UL>
94      *
95      * @return a new instance of a MessageFactory
96      *
97      * @exception SOAPException if there was an error in creating the
98      *         default implementation of the
99      *         MessageFactory.
100     * @see SAAJMetaFactory
101     */
102     public static MessageFactory newInstance() throws SOAPException {
103
104         try {
105             MessageFactory factory = (MessageFactory) FactoryFinder.find(
106                 MESSAGE_FACTORY_PROPERTY,
107                 DEFAULT_MESSAGE_FACTORY,
108                 false);
109
110             if (factory != null) {
111                 return factory;
112             }
113             return newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);
114         } catch (Exception ex) {
115             throw new SOAPException(
116                 "Unable to create message factory for SOAP: "
117                 + ex.getMessage());
118         }
119     }

```

```

117     }
118
119 }
120
121 /**
122  * Creates a new MessageFactory object that is an instance
123  * of the specified implementation. May be a dynamic message factory,
124  * a SOAP 1.1 message factory, or a SOAP 1.2 message factory. A dynamic
125  * message factory creates messages based on the MIME headers specified
126  * as arguments to the createMessage method.
127  *
128  * This method uses the SAAJMetaFactory to locate the implementation class
129  * and create the MessageFactory instance.
130  *
131  * @return a new instance of a MessageFactory
132  *
133  * @param protocol a string constant representing the class of the
134  *                specified message factory implementation. May be
135  *                either DYNAMIC_SOAP_PROTOCOL,
136  *                DEFAULT_SOAP_PROTOCOL (which is the same
137  *                as) SOAP_1_1_PROTOCOL, or
138  *                SOAP_1_2_PROTOCOL.
139  *
140  * @exception SOAPException if there was an error in creating the
141  *                specified implementation of MessageFactory.
142  * @see SAAJMetaFactory
143  * @since SAAJ 1.3
144  */
145 public static MessageFactory newInstance(String protocol) throws SOAPException {
146     return SAAJMetaFactory.getInstance().newMessageFactory(protocol);
147 }
148
149 /**
150  * Creates a new SOAPMessage object with the default
151  * SOAPPart, SOAPEnvelope, SOAPBody,
152  * and SOAPHeader objects. Profile-specific message factories
153  * can choose to prepopulate the SOAPMessage object with
154  * profile-specific headers.
155  * <P>
156  * Content can be added to this message's SOAPPart object, and
157  * the message can be sent "as is" when a message containing only a SOAP part
158  * is sufficient. Otherwise, the SOAPMessage object needs
159  * to create one or more AttachmentPart objects and
160  * add them to itself. Any content that is not in XML format must be
161  * in an AttachmentPart object.
162  *
163  * @return a new SOAPMessage object
164  * @exception SOAPException if a SOAP error occurs
165  * @exception UnsupportedOperationException if the protocol of this
166  *                MessageFactory instance is DYNAMIC_SOAP_PROTOCOL
167  */
168 public abstract SOAPMessage createMessage()
169     throws SOAPException;

```

```

170
171 /**
172  * Internalizes the contents of the given <code>InputStream</code> object into a
173  * new <code>SOAPMessage</code> object and returns the <code>SOAPMessage</code>
174  * object.
175  *
176  * @param in the <code>InputStream</code> object that contains the data
177  *           for a message
178  * @param headers the transport-specific headers passed to the
179  *               message in a transport-independent fashion for creation of the
180  *               message
181  * @return a new <code>SOAPMessage</code> object containing the data from
182  *         the given <code>InputStream</code> object
183  *
184  * @exception IOException if there is a problem in reading data from
185  *                       the input stream
186  *
187  * @exception SOAPException may be thrown if the message is invalid
188  *
189  * @exception IllegalArgumentException if the <code>MessageFactory</code>
190  *       requires one or more MIME headers to be present in the
191  *       <code>headers</code> parameter and they are missing.
192  *       <code>MessageFactory</code> implementations for
193  *       <code>SOAP_1_1_PROTOCOL</code> or
194  *       <code>SOAP_1_2_PROTOCOL</code> must not throw
195  *       <code>IllegalArgumentException</code> for this reason.
196  */
197 public abstract SOAPMessage createMessage(MimeHeaders headers,
198                                         InputStream in)
199     throws IOException, SOAPException;
200 }

```

## 17.6 MimeHeader.java

Secuencia 167: javax/xml/soap/MimeHeader.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *

```

```
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.soap;
27
28
29 /**
30  * An object that stores a MIME header name and its value. One or more
31  * MimeHeader objects may be contained in a MimeHeaders
32  * object.
33  *
34  * @see MimeHeaders
35  */
36 public class MimeHeader {
37
38     private String name;
39     private String value;
40
41     /**
42      * Constructs a MimeHeader object initialized with the given
43      * name and value.
44      *
45      * @param name a String giving the name of the header
46      * @param value a String giving the value of the header
47      */
48     public MimeHeader(String name, String value) {
49         this.name = name;
50         this.value = value;
51     }
52
53     /**
54      * Returns the name of this MimeHeader object.
55      *
56      * @return the name of the header as a String
57      */
58     public String getName() {
59         return name;
60     }
61
62     /**
63      * Returns the value of this MimeHeader object.
64      *
65      * @return the value of the header as a String
66      */
67     public String getValue() {
68         return value;
69     }
70 }
```

## 17.7 MimeHeaders.java

Secuencia 168: javax/xml/soap/MimeHeaders.java

```
1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import java.util.Iterator;
29 import java.util.Vector;
30
31 /**
32 * A container for MimeHeader objects, which represent
33 * the MIME headers present in a MIME part of a message.
34 *
35 * <p>This class is used primarily when an application wants to
36 * retrieve specific attachments based on certain MIME headers and
37 * values. This class will most likely be used by implementations of
38 * AttachmentPart and other MIME dependent parts of the SAAJ
39 * API.
40 * @see SOAPMessage#getAttachments
41 * @see AttachmentPart
42 */
43 public class MimeHeaders {
44     private Vector headers;
45
46     /**
47     * Constructs a default MimeHeaders object initialized with
48     * an empty Vector object.
49     */
50     public MimeHeaders() {
```

```

51     headers = new Vector();
52 }
53
54 /**
55  * Returns all of the values for the specified header as an array of
56  * <code>String</code> objects.
57  *
58  * @param   name the name of the header for which values will be returned
59  * @return  a <code>String</code> array with all of the values for the
60  *         specified header
61  * @see    #setHeader
62  */
63 public String[] getHeader(String name) {
64     Vector values = new Vector();
65
66     for(int i = 0; i < headers.size(); i++) {
67         MimeHeader hdr = (MimeHeader) headers.elementAt(i);
68         if (hdr.getName().equalsIgnoreCase(name)
69             && hdr.getValue() != null)
70             values.addElement(hdr.getValue());
71     }
72
73     if (values.size() == 0)
74         return null;
75
76     String r[] = new String[values.size()];
77     values.copyInto(r);
78     return r;
79 }
80
81 /**
82  * Replaces the current value of the first header entry whose name matches
83  * the given name with the given value, adding a new header if no existing header
84  * name matches. This method also removes all matching headers after the first one.
85  * <P>
86  * Note that RFC822 headers can contain only US-ASCII characters.
87  *
88  * @param   name a <code>String</code> with the name of the header for
89  *           which to search
90  * @param   value a <code>String</code> with the value that will replace the
91  *           current value of the specified header
92  *
93  * @exception IllegalArgumentException if there was a problem in the
94  *         mime header name or the value being set
95  * @see    #getHeader
96  */
97 public void setHeader(String name, String value)
98 {
99     boolean found = false;
100
101     if ((name == null) || name.equals(""))
102         throw new IllegalArgumentException("Illegal MimeHeader name");
103

```



```

104     for(int i = 0; i < headers.size(); i++) {
105         MimeHeader hdr = (MimeHeader) headers.elementAt(i);
106         if (hdr.getName().equalsIgnoreCase(name)) {
107             if (!found) {
108                 headers.setElementAt(new MimeHeader(hdr.getName(),
109                                                     value), i);
110                 found = true;
111             }
112             else
113                 headers.removeElementAt(i--);
114         }
115     }
116
117     if (!found)
118         addHeader(name, value);
119 }
120
121 /**
122  * Adds a <code>MimeHeader</code> object with the specified name and value
123  * to this <code>MimeHeaders</code> object's list of headers.
124  * <P>
125  * Note that RFC822 headers can contain only US-ASCII characters.
126  *
127  * @param   name a <code>String</code> with the name of the header to
128  *           be added
129  * @param   value a <code>String</code> with the value of the header to
130  *           be added
131  *
132  * @exception IllegalArgumentException if there was a problem in the
133  * mime header name or value being added
134  */
135 public void addHeader(String name, String value)
136 {
137     if ((name == null) || name.equals(""))
138         throw new IllegalArgumentException("Illegal MimeHeader name");
139
140     int pos = headers.size();
141
142     for(int i = pos - 1 ; i >= 0; i--) {
143         MimeHeader hdr = (MimeHeader) headers.elementAt(i);
144         if (hdr.getName().equalsIgnoreCase(name)) {
145             headers.insertElementAt(new MimeHeader(name, value),
146                                     i+1);
147             return;
148         }
149     }
150     headers.addElement(new MimeHeader(name, value));
151 }
152
153 /**
154  * Remove all <code>MimeHeader</code> objects whose name matches the
155  * given name.
156  *

```

```
157     * @param   name a <code>String</code> with the name of the header for
158     *           which to search
159     */
160     public void removeHeader(String name) {
161         for(int i = 0; i < headers.size(); i++) {
162             MimeHeader hdr = (MimeHeader) headers.elementAt(i);
163             if (hdr.getName().equalsIgnoreCase(name))
164                 headers.removeElementAt(i--);
165         }
166     }
167
168     /**
169     * Removes all the header entries from this <code>MimeHeaders</code> object.
170     */
171     public void removeAllHeaders() {
172         headers.removeAllElements();
173     }
174
175
176     /**
177     * Returns all the <code>MimeHeader</code>s in this <code>MimeHeaders</code> object.
178     *
179     * @return   an <code>Iterator</code> object over this <code>MimeHeaders</code>
180     *           object's list of <code>MimeHeader</code> objects
181     */
182     public Iterator getAllHeaders() {
183         return headers.iterator();
184     }
185
186     class MatchingIterator implements Iterator {
187         private boolean match;
188         private Iterator iterator;
189         private String[] names;
190         private Object nextHeader;
191
192         MatchingIterator(String[] names, boolean match) {
193             this.match = match;
194             this.names = names;
195             this.iterator = headers.iterator();
196         }
197
198         private Object nextMatch() {
199             next:
200                 while (iterator.hasNext()) {
201                     MimeHeader hdr = (MimeHeader) iterator.next();
202
203                     if (names == null)
204                         return match ? null : hdr;
205
206                     for(int i = 0; i < names.length; i++)
207                         if (hdr.getName().equalsIgnoreCase(names[i]))
208                             if (match)
209                                 return hdr;
```

```

210         else
211             continue next;
212         if (!match)
213             return hdr;
214     }
215     return null;
216 }
217
218
219 public boolean hasNext() {
220     if (nextHeader == null)
221         nextHeader = nextMatch();
222     return nextHeader != null;
223 }
224
225 public Object next() {
226     // hasNext should've prefetched the header for us,
227     // return it.
228     if (nextHeader != null) {
229         Object ret = nextHeader;
230         nextHeader = null;
231         return ret;
232     }
233     if (hasNext())
234         return nextHeader;
235     return null;
236 }
237
238 public void remove() {
239     iterator.remove();
240 }
241 }
242
243
244 /**
245  * Returns all the <code>MimeHeader</code> objects whose name matches
246  * a name in the given array of names.
247  *
248  * @param names an array of <code>String</code> objects with the names
249  *         for which to search
250  * @return an <code>Iterator</code> object over the <code>MimeHeader</code>
251  *         objects whose name matches one of the names in the given list
252  */
253 public Iterator getMatchingHeaders(String[] names) {
254     return new MatchingIterator(names, true);
255 }
256
257 /**
258  * Returns all of the <code>MimeHeader</code> objects whose name does not
259  * match a name in the given array of names.
260  *
261  * @param names an array of <code>String</code> objects with the names
262  *         for which to search

```

```

263     * @return an Iterator object over the MimeHeader
264     *         objects whose name does not match one of the names in the given list
265     */
266     public Iterator getNonMatchingHeaders(String[] names) {
267         return new MatchingIterator(names, false);
268     }
269 }

```

## 17.8 Name.java

Secuencia 169: javax/xml/soap/Name.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 /**
29  * A representation of an XML name. This interface provides methods for
30  * getting the local and namespace-qualified names and also for getting the
31  * prefix associated with the namespace for the name. It is also possible
32  * to get the URI of the namespace.
33  * <P>
34  * The following is an example of a namespace declaration in an element.
35  * <PRE>
36  * <?wombat:GetLastTradePrice xmlns:wombat="http://www.wombat.org/trader">
37  * </PRE>
38  * ("xmlns" stands for "XML namespace".)
39  * The following
40  * shows what the methods in the Name interface will return.
41  * <UL>
42  * <LI><code>getQualifiedName</code> will return "prefix:LocalName" =

```

```

43 *      "WOMBAT:GetLastTradePrice"
44 * <LI><code>getURI</code> will return "http://www.wombat.org/trader"
45 * <LI><code>getLocalName</code> will return "GetLastTracePrice"
46 * <LI><code>getPrefix</code> will return "WOMBAT"
47 * </UL>
48 * <P>
49 * XML namespaces are used to disambiguate SOAP identifiers from
50 * application-specific identifiers.
51 * <P>
52 * <code>Name</code> objects are created using the method
53 * <code>SOAPEnvelope.createName</code>, which has two versions.
54 * One method creates <code>Name</code> objects with
55 * a local name, a namespace prefix, and a namespace URI.
56 * and the second creates <code>Name</code> objects with just a local name.
57 * The following line of
58 * code, in which <i>se</i> is a <code>SOAPEnvelope</code> object, creates a new
59 * <code>Name</code> object with all three.
60 * <PRE>
61 *     Name name = se.createName("GetLastTradePrice", "WOMBAT",
62 *                               "http://www.wombat.org/trader");
63 * </PRE>
64 * The following line of code gives an example of how a <code>Name</code> object
65 * can be used. The variable <i>element</i> is a <code>SOAPElement</code> object.
66 * This code creates a new <code>SOAPElement</code> object with the given name and
67 * adds it to <i>element</i>.
68 * <PRE>
69 *     element.addChildElement(name);
70 * </PRE>
71 * <P>
72 * The <code>Name</code> interface may be deprecated in a future release of SAAJ
73 * in favor of <code>javax.xml.namespace.QName</code>
74 * @see SOAPEnvelope#createName(String, String, String) SOAPEnvelope.createName
75 * @see SOAPFactory#createName(String, String, String) SOAPFactory.createName
76 */
77 public interface Name {
78     /**
79      * Gets the local name part of the XML name that this <code>Name</code>
80      * object represents.
81      *
82      * @return a string giving the local name
83      */
84     String getLocalName();
85
86     /**
87      * Gets the namespace-qualified name of the XML name that this
88      * <code>Name</code> object represents.
89      *
90      * @return the namespace-qualified name as a string
91      */
92     String getQualifiedName();
93
94     /**
95      * Returns the prefix that was specified when this <code>Name</code> object

```

```

96     * was initialized. This prefix is associated with the namespace for the XML
97     * name that this <code>Name</code> object represents.
98     *
99     * @return the prefix as a string
100    */
101    String getPrefix();
102
103    /**
104     * Returns the URI of the namespace for the XML
105     * name that this <code>Name</code> object represents.
106     *
107     * @return the URI as a string
108     */
109    String getURI();
110 }

```

## 17.9 Node.java

Secuencia 170: javax/xml/soap/Node.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 /**
29  * A representation of a node (element) in an XML document.
30  * This interface extends the standard DOM Node interface with methods for
31  * getting and setting the value of a node, for
32  * getting and setting the parent of a node, and for removing a node.
33  */
34 public interface Node extends org.w3c.dom.Node {

```

```

35  /**
36   * Returns the value of this node if this is a <code>Text</code> node or the
37   * value of the immediate child of this node otherwise.
38   * If there is an immediate child of this <code>Node</code> that it is a
39   * <code>Text</code> node then it's value will be returned. If there is
40   * more than one <code>Text</code> node then the value of the first
41   * <code>Text</code> Node will be returned.
42   * Otherwise <code>null</code> is returned.
43   *
44   * @return a <code>String</code> with the text of this node if this is a
45   *         <code>Text</code> node or the text contained by the first
46   *         immediate child of this <code>Node</code> object that is a
47   *         <code>Text</code> object if such a child exists;
48   *         <code>null</code> otherwise.
49   */
50  public String getValue();
51
52  /**
53   * If this is a Text node then this method will set its value,
54   * otherwise it sets the value of the immediate (Text) child of this node.
55   * The value of the immediate child of this node can be set only if, there is
56   * one child node and that node is a <code>Text</code> node, or if
57   * there are no children in which case a child <code>Text</code> node will be
58   * created.
59   *
60   * @exception IllegalStateException if the node is not a <code>Text</code>
61   *         node and either has more than one child node or has a child
62   *         node that is not a <code>Text</code> node.
63   *
64   * @since SAAJ 1.2
65   */
66  public void setValue(String value);
67
68  /**
69   * Sets the parent of this <code>Node</code> object to the given
70   * <code>SOAPElement</code> object.
71   *
72   * @param parent the <code>SOAPElement</code> object to be set as
73   *         the parent of this <code>Node</code> object
74   *
75   * @exception SOAPException if there is a problem in setting the
76   *         parent to the given element
77   * @see #getParentElement
78   */
79  public void setParentElement(SOAPElement parent) throws SOAPException;
80
81  /**
82   * Returns the parent element of this <code>Node</code> object.
83   * This method can throw an <code>UnsupportedOperationException</code>
84   * if the tree is not kept in memory.
85   *
86   * @return the <code>SOAPElement</code> object that is the parent of
87   *         this <code>Node</code> object or <code>null</code> if this

```

```

88      *      <code>Node</code> object is root
89      *
90      * @exception UnsupportedOperationException if the whole tree is not
91      *      kept in memory
92      * @see #setParentElement
93      */
94      public SOAPElement getParentElement();
95
96      /**
97      * Removes this <code>Node</code> object from the tree.
98      */
99      public void detachNode();
100
101      /**
102      * Notifies the implementation that this <code>Node</code>
103      * object is no longer being used by the application and that the
104      * implementation is free to reuse this object for nodes that may
105      * be created later.
106      * <P>
107      * Calling the method <code>recycleNode</code> implies that the method
108      * <code>detachNode</code> has been called previously.
109      */
110      public void recycleNode();
111
112 }

```

## 17.10 SAAJMetaFactory.java

Secuencia 171: javax/xml/soap/SAAJMetaFactory.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */

```



```

25
26 package javax.xml.soap;
27
28 /**
29  * The access point for the implementation classes of the factories defined in the
30  * SAAJ API. All of the <code>newInstance</code> methods defined on factories in
31  * SAAJ 1.3 defer to instances of this class to do the actual object creation.
32  * The implementations of <code>newInstance</code> methods (in SOAPFactory and MessageFactory)
33  * that existed in SAAJ 1.2 have been updated to also delegate to the SAAJMetaFactory when the SAAJ
34  * 1.2
35  * defined lookup fails to locate the Factory implementation class name.
36  *
37  * <p>
38  * SAAJMetaFactory is a service provider interface. There are no public methods on this
39  * class.
40  *
41  * @author SAAJ RI Development Team
42  * @since SAAJ 1.3
43  */
44 public abstract class SAAJMetaFactory {
45     static private final String META_FACTORY_CLASS_PROPERTY =
46         "javax.xml.soap.MetaFactory";
47     static final String DEFAULT_META_FACTORY_CLASS =
48         "com.sun.xml.internal.messaging.saaj.soap.SAAJMetaFactoryImpl";
49
50     /**
51      * Creates a new instance of a concrete <code>SAAJMetaFactory</code> object.
52      * The SAAJMetaFactory is an SPI, it pulls the creation of the other factories together into a
53      * single place. Changing out the SAAJMetaFactory has the effect of changing out the entire
54      * SAAJ
55      * implementation. Service providers provide the name of their <code>SAAJMetaFactory</code>
56      * implementation.
57      *
58      * This method uses the following ordered lookup procedure to determine the SAAJMetaFactory
59      * implementation class to load:
60      *
61      * <UL>
62      * <LI> Use the javax.xml.soap.MetaFactory system property.
63      * <LI> Use the properties file "lib/jaxm.properties" in the JRE directory. This configuration
64      * file is in standard
65      * java.util.Properties format and contains the fully qualified name of the implementation
66      * class with the key being the
67      * system property defined above.
68      * <LI> Use the Services API (as detailed in the JAR specification), if available, to
69      * determine the classname. The Services API
70      * will look for a classname in the file META-INF/services/javax.xml.soap.MetaFactory in jars
71      * available to the runtime.
72      * <LI> Default to com.sun.xml.internal.messaging.saaj.soap.SAAJMetaFactoryImpl.
73      * </UL>
74      *
75      * @return a concrete <code>SAAJMetaFactory</code> object
76      * @exception SOAPException if there is an error in creating the <code>SAAJMetaFactory</code>
77      */

```

```

71     static SAAJMetaFactory getInstance() throws SOAPException {
72         try {
73             SAAJMetaFactory instance =
74                 (SAAJMetaFactory) FactoryFinder.find(
75                     META_FACTORY_CLASS_PROPERTY,
76                     DEFAULT_META_FACTORY_CLASS);
77             return instance;
78         } catch (Exception e) {
79             throw new SOAPException(
80                 "Unable to create SAAJ meta-factory" + e.getMessage());
81         }
82     }
83
84     protected SAAJMetaFactory() { }
85
86     /**
87      * Creates a <code>MessageFactory</code> object for
88      * the given <code>String</code> protocol.
89      *
90      * @param protocol a <code>String</code> indicating the protocol
91      * @exception SOAPException if there is an error in creating the
92      *         MessageFactory
93      * @see SOAPConstants#SOAP_1_1_PROTOCOL
94      * @see SOAPConstants#SOAP_1_2_PROTOCOL
95      * @see SOAPConstants#DYNAMIC_SOAP_PROTOCOL
96      */
97     protected abstract MessageFactory newMessageFactory(String protocol)
98         throws SOAPException;
99
100    /**
101     * Creates a <code>SOAPFactory</code> object for
102     * the given <code>String</code> protocol.
103     *
104     * @param protocol a <code>String</code> indicating the protocol
105     * @exception SOAPException if there is an error in creating the
106     *         SOAPFactory
107     * @see SOAPConstants#SOAP_1_1_PROTOCOL
108     * @see SOAPConstants#SOAP_1_2_PROTOCOL
109     * @see SOAPConstants#DYNAMIC_SOAP_PROTOCOL
110     */
111     protected abstract SOAPFactory newSOAPFactory(String protocol)
112         throws SOAPException;
113 }

```

## 17.11 SAAJResult.java

Secuencia 172: javax/xml/soap/SAAJResult.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import javax.xml.transform.dom.DOMResult;
29
30 /**
31  * Acts as a holder for the results of a JAXP transformation or a JAXB
32  * marshalling, in the form of a SAAJ tree. These results should be accessed
33  * by using the {@link #getResult()} method. The {@link DOMResult#getNode()}
34  * method should be avoided in almost all cases.
35  *
36  * @author XWS-Security Development Team
37  *
38  * @since SAAJ 1.3
39  */
40 public class SAAJResult extends DOMResult {
41
42     /**
43      * Creates a SAAJResult that will present results in the form
44      * of a SAAJ tree that supports the default (SOAP 1.1) protocol.
45      * <p>
46      * This kind of SAAJResult is meant for use in situations where the
47      * results will be used as a parameter to a method that takes a parameter
48      * whose type, such as SOAPElement, is drawn from the SAAJ
49      * API. When used in a transformation, the results are populated into the
50      * SOAPPart of a SOAPMessage that is created internally.
51      * The SOAPPart returned by {@link DOMResult#getNode()}
52      * is not guaranteed to be well-formed.
53      *
54      * @throws SOAPException if there is a problem creating a SOAPMessage
55      *
56      * @since SAAJ 1.3
57      */
58     public SAAJResult() throws SOAPException {
59         this(MessageFactory.newInstance().createMessage());
60     }
61 }
```

```

60     }
61
62     /**
63      * Creates a SAAJResult that will present results in the form
64      * of a SAAJ tree that supports the specified protocol. The
65      * DYNAMIC_SOAP_PROTOCOL is ambiguous in this context and will
66      * cause this constructor to throw an UnsupportedOperationException.
67      * <p>
68      * This kind of SAAJResult is meant for use in situations where the
69      * results will be used as a parameter to a method that takes a parameter
70      * whose type, such as SOAPElement, is drawn from the SAAJ
71      * API. When used in a transformation the results are populated into the
72      * SOAPPart of a SOAPMessage that is created
73      * internally. The SOAPPart returned by {@link DOMResult#getNode()}
74      * is not guaranteed to be well-formed.
75      *
76      * @param protocol - the name of the SOAP protocol that the resulting SAAJ
77      *                  tree should support
78      *
79      * @throws SOAPException if a SOAPMessage supporting the
80      *                  specified protocol cannot be created
81      *
82      * @since SAAJ 1.3
83      */
84     public SAAJResult(String protocol) throws SOAPException {
85         this(MessageFactory.newInstance(protocol).createMessage());
86     }
87
88     /**
89      * Creates a SAAJResult that will write the results into the
90      * SOAPPart of the supplied SOAPMessage.
91      * In the normal case these results will be written using DOM APIs and,
92      * as a result, the finished SOAPPart will not be guaranteed
93      * to be well-formed unless the data used to create it is also well formed.
94      * When used in a transformation the validity of the SOAPMessage
95      * after the transformation can be guaranteed only by means outside SAAJ
96      * specification.
97      *
98      * @param message - the message whose SOAPPart will be
99      *                  populated as a result of some transformation or
100     *                  marshalling operation
101     *
102     * @since SAAJ 1.3
103     */
104     public SAAJResult(SOAPMessage message) {
105         super(message.getSOAPPart());
106     }
107
108     /**
109      * Creates a SAAJResult that will write the results as a
110      * child node of the SOAPElement specified. In the normal
111      * case these results will be written using DOM APIs and as a result may
112      * invalidate the structure of the SAAJ tree. This kind of

```

```

113     * <code>SAAJResult</code> should only be used when the validity of the
114     * incoming data can be guaranteed by means outside of the SAAJ
115     * specification.
116     *
117     * @param rootNode - the root to which the results will be appended
118     *
119     * @since SAAJ 1.3
120     */
121     public SAAJResult(SOAPElement rootNode) {
122         super(rootNode);
123     }
124
125
126     /**
127     * @return the resulting Tree that was created under the specified root Node.
128     * @since SAAJ 1.3
129     */
130     public javax.xml.soap.Node getResult() {
131         return (javax.xml.soap.Node)super.getNode().getFirstChild();
132     }
133 }

```

## 17.12 SOAPBody.java

Secuencia 173: javax/xml/soap/SOAPBody.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import java.util.Locale;

```

```

29
30 import org.w3c.dom.Document;
31
32 import javax.xml.namespace.QName;
33
34 /**
35  * An object that represents the contents of the SOAP body
36  * element in a SOAP message. A SOAP body element consists of XML data
37  * that affects the way the application-specific content is processed.
38  * <P>
39  * A <code>SOAPBody</code> object contains <code>SOAPBodyElement</code>
40  * objects, which have the content for the SOAP body.
41  * A <code>SOAPFault</code> object, which carries status and/or
42  * error information, is an example of a <code>SOAPBodyElement</code> object.
43  *
44  * @see SOAPFault
45  */
46 public interface SOAPBody extends SOAPElement {
47
48     /**
49      * Creates a new <code>SOAPFault</code> object and adds it to
50      * this <code>SOAPBody</code> object. The new <code>SOAPFault</code> will
51      * have default values set for the mandatory child elements. The type of
52      * the <code>SOAPFault</code> will be a SOAP 1.1 or a SOAP 1.2 <code>SOAPFault</code>
53      * depending on the <code>protocol</code> specified while creating the
54      * <code>MessageFactory</code> instance.
55      * <p>
56      * A <code>SOAPBody</code> may contain at most one <code>SOAPFault</code>
57      * child element.
58      *
59      * @return the new <code>SOAPFault</code> object
60      * @exception SOAPException if there is a SOAP error
61      */
62     public SOAPFault addFault() throws SOAPException;
63
64
65     /**
66      * Creates a new <code>SOAPFault</code> object and adds it to
67      * this <code>SOAPBody</code> object. The type of the
68      * <code>SOAPFault</code> will be a SOAP 1.1 or a SOAP 1.2
69      * <code>SOAPFault</code> depending on the <code>protocol</code>
70      * specified while creating the <code>MessageFactory</code> instance.
71      * <p>
72      * For SOAP 1.2 the <code>faultCode</code> parameter is the value of the
73      * <i>Fault/Code/Value</i> element and the <code>faultString</code> parameter
74      * is the value of the <i>Fault/Reason/Text</i> element. For SOAP 1.1
75      * the <code>faultCode</code> parameter is the value of the <code>faultcode</code>
76      * element and the <code>faultString</code> parameter is the value of the <code>faultstring</code>
77      * element.
78      * <p>
79      * A <code>SOAPBody</code> may contain at most one <code>SOAPFault</code>
80      * child element.

```

```

81      *
82      * @param faultCode a <code>Name</code> object giving the fault
83      *      code to be set; must be one of the fault codes defined in the Version
84      *      of SOAP specification in use
85      * @param faultString a <code>String</code> giving an explanation of
86      *      the fault
87      * @param locale a {@link java.util.Locale} object indicating
88      *      the native language of the <code>faultString</code>
89      * @return the new <code>SOAPFault</code> object
90      * @exception SOAPException if there is a SOAP error
91      * @see SOAPFault#setFaultCode
92      * @see SOAPFault#setFaultString
93      * @since SAAJ 1.2
94      */
95      public SOAPFault addFault(Name faultCode, String faultString, Locale locale) throws
          SOAPException;
96
97      /**
98      * Creates a new <code>SOAPFault</code> object and adds it to this
99      * <code>SOAPBody</code> object. The type of the <code>SOAPFault</code>
100     * will be a SOAP 1.1 or a SOAP 1.2 <code>SOAPFault</code> depending on
101     * the <code>protocol</code> specified while creating the <code>MessageFactory</code>
102     * instance.
103     * <p>
104     * For SOAP 1.2 the <code>faultCode</code> parameter is the value of the
105     * <i>Fault/Code/Value</i> element and the <code>faultString</code> parameter
106     * is the value of the <i>Fault/Reason/Text</i> element. For SOAP 1.1
107     * the <code>faultCode</code> parameter is the value of the <code>faultcode</code>
108     * element and the <code>faultString</code> parameter is the value of the <code>faultstring</code>
109     * element.
110     * <p>
111     * A <code>SOAPBody</code> may contain at most one <code>SOAPFault</code>
112     * child element.
113     *
114     * @param faultCode
115     *      a <code>QName</code> object giving the fault code to be
116     *      set; must be one of the fault codes defined in the version
117     *      of SOAP specification in use.
118     * @param faultString
119     *      a <code>String</code> giving an explanation of the fault
120     * @param locale
121     *      a {@link java.util.Locale Locale} object indicating the
122     *      native language of the <code>faultString</code>
123     * @return the new <code>SOAPFault</code> object
124     * @exception SOAPException
125     *      if there is a SOAP error
126     * @see SOAPFault#setFaultCode
127     * @see SOAPFault#setFaultString
128     * @see SOAPBody#addFault(Name faultCode, String faultString, Locale locale)
129     *
130     * @since SAAJ 1.3
131     */

```

```

132 public SOAPFault addFault(QName faultCode, String faultString, Locale locale)
133     throws SOAPException;
134
135 /**
136  * Creates a new SOAPFault object and adds it to this
137  * SOAPBody object. The type of the SOAPFault
138  * will be a SOAP 1.1 or a SOAP 1.2 SOAPFault depending on
139  * the protocol specified while creating the MessageFactory
140  * instance.
141  * <p>
142  * For SOAP 1.2 the faultCode parameter is the value of the
143  * <i>Fault/Code/Value</i> element and the faultString parameter
144  * is the value of the <i>Fault/Reason/Text</i> element. For SOAP 1.1
145  * the faultCode parameter is the value of the <i>faultcode</i>
146  * element and the faultString parameter is the value of the <i>faultstring</i>
147  * element.
148  * <p>
149  * In case of a SOAP 1.2 fault, the default value for the mandatory xml:lang
150  * attribute on the <i>Fault/Reason/Text</i> element will be set to
151  * java.util.Locale.getDefault()
152  * <p>
153  * A SOAPBody may contain at most one SOAPFault
154  * child element.
155  *
156  * @param faultCode
157  *     a Name object giving the fault code to be set;
158  *     must be one of the fault codes defined in the version of SOAP
159  *     specification in use
160  * @param faultString
161  *     a String giving an explanation of the fault
162  * @return the new SOAPFault object
163  * @exception SOAPException
164  *     if there is a SOAP error
165  * @see SOAPFault#setFaultCode
166  * @see SOAPFault#setFaultString
167  * @since SAAJ 1.2
168  */
169 public SOAPFault addFault(Name faultCode, String faultString)
170     throws SOAPException;
171
172 /**
173  * Creates a new SOAPFault object and adds it to this SOAPBody
174  * object. The type of the SOAPFault
175  * will be a SOAP 1.1 or a SOAP 1.2 SOAPFault depending on
176  * the protocol specified while creating the MessageFactory
177  * instance.
178  * <p>
179  * For SOAP 1.2 the faultCode parameter is the value of the
180  * <i>Fault/Code/Value</i> element and the faultString parameter
181  * is the value of the <i>Fault/Reason/Text</i> element. For SOAP 1.1
182  * the faultCode parameter is the value of the <i>faultcode</i>
183  * element and the faultString parameter is the value of the <i>faultstring</i>
184  * element.

```



```

185 * <p>
186 * In case of a SOAP 1.2 fault, the default value for the mandatory <code>xml:lang</code>
187 * attribute on the <i>Fault/Reason/Text</i> element will be set to
188 * <code>java.util.Locale.getDefault()</code>
189 * <p>
190 * A <code>SOAPBody</code> may contain at most one <code>SOAPFault</code>
191 * child element
192 *
193 * @param faultCode
194 *     a <code>QName</code> object giving the fault code to be
195 *     set; must be one of the fault codes defined in the version
196 *     of SOAP specification in use
197 * @param faultString
198 *     a <code>String</code> giving an explanation of the fault
199 * @return the new <code>SOAPFault</code> object
200 * @exception SOAPException
201 *     if there is a SOAP error
202 * @see SOAPFault#setFaultCode
203 * @see SOAPFault#setFaultString
204 * @see SOAPBody#addFault(Name faultCode, String faultString)
205 * @since SAAJ 1.3
206 */
207 public SOAPFault addFault(QName faultCode, String faultString)
208     throws SOAPException;
209
210 /**
211 * Indicates whether a <code>SOAPFault</code> object exists in this
212 * <code>SOAPBody</code> object.
213 *
214 * @return <code>true</code> if a <code>SOAPFault</code> object exists
215 *     in this <code>SOAPBody</code> object; <code>false</code>
216 *     otherwise
217 */
218 public boolean hasFault();
219
220 /**
221 * Returns the <code>SOAPFault</code> object in this <code>SOAPBody</code>
222 * object.
223 *
224 * @return the <code>SOAPFault</code> object in this <code>SOAPBody</code>
225 *     object if present, null otherwise.
226 */
227 public SOAPFault getFault();
228
229 /**
230 * Creates a new <code>SOAPBodyElement</code> object with the specified
231 * name and adds it to this <code>SOAPBody</code> object.
232 *
233 * @param name
234 *     a <code>Name</code> object with the name for the new <code>SOAPBodyElement</code>
235 *     object
236 * @return the new <code>SOAPBodyElement</code> object
237 * @exception SOAPException

```

```

238     *           if a SOAP error occurs
239     * @see SOAPBody#addBodyElement(javax.xml.namespace.QName)
240     */
241     public SOAPBodyElement addBodyElement(Name name) throws SOAPException;
242
243
244     /**
245     * Creates a new SOAPBodyElement object with the specified
246     * QName and adds it to this SOAPBody object.
247     *
248     * @param qname
249     *         a QName object with the qname for the new
250     *         SOAPBodyElement object
251     * @return the new SOAPBodyElement object
252     * @exception SOAPException
253     *         if a SOAP error occurs
254     * @see SOAPBody#addBodyElement(Name)
255     * @since SAAJ 1.3
256     */
257     public SOAPBodyElement addBodyElement(QName qname) throws SOAPException;
258
259     /**
260     * Adds the root node of the DOM {@link org.w3c.dom.Document}
261     * to this SOAPBody object.
262     * <p>
263     * Calling this method invalidates the document parameter.
264     * The client application should discard all references to this Document
265     * and its contents upon calling addDocument. The behavior
266     * of an application that continues to use such references is undefined.
267     *
268     * @param document
269     *         the Document object whose root node will be
270     *         added to this SOAPBody.
271     * @return the SOAPBodyElement that represents the root node
272     *         that was added.
273     * @exception SOAPException
274     *         if the Document cannot be added
275     * @since SAAJ 1.2
276     */
277     public SOAPBodyElement addDocument(org.w3c.dom.Document document)
278         throws SOAPException;
279
280     /**
281     * Creates a new DOM {@link org.w3c.dom.Document} and sets
282     * the first child of this SOAPBody as it's document
283     * element. The child SOAPElement is removed as part of the
284     * process.
285     *
286     * @return the {@link org.w3c.dom.Document} representation
287     *         of the SOAPBody content.
288     *
289     * @exception SOAPException
290     *         if there is not exactly one child SOAPElement of the 

```

```

291     *          <code>SOAPBody</code>.
292     *
293     * @since SAAJ 1.3
294     */
295     public org.w3c.dom.Document extractContentAsDocument()
296         throws SOAPException;
297 }

```

### 17.13 SOAPBodyElement.java

Secuencia 174: javax/xml/soap/SOAPBodyElement.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 /**
29  * A <code>SOAPBodyElement</code> object represents the contents in
30  * a <code>SOAPBody</code> object. The <code>SOAPFault</code> interface
31  * is a <code>SOAPBodyElement</code> object that has been defined.
32  * <P>
33  * A new <code>SOAPBodyElement</code> object can be created and added
34  * to a <code>SOAPBody</code> object with the <code>SOAPBody</code>
35  * method <code>addBodyElement</code>. In the following line of code,
36  * <code>sb</code> is a <code>SOAPBody</code> object, and
37  * <code>myName</code> is a <code>Name</code> object.
38  * <PRE>
39  *     SOAPBodyElement sbe = sb.addBodyElement(myName);
40  * </PRE>
41  */
42 public interface SOAPBodyElement extends SOAPElement {

```

43 }

## 17.14 SOAPConnection.java

Secuencia 175: javax/xml/soap/SOAPConnection.java

```
1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28
29 /**
30 * A point-to-point connection that a client can use for sending messages
31 * directly to a remote party (represented by a URL, for instance).
32 * <p>
33 * The SOAPConnection class is optional. Some implementations may
34 * not implement this interface in which case the call to
35 * <code>SOAPConnectionFactory.newInstance()</code> (see below) will
36 * throw an <code>UnsupportedOperationException</code>.
37 * <p>
38 * A client can obtain a <code>SOAPConnection</code> object using a
39 * {@link SOAPConnectionFactory} object as in the following example:
40 * <PRE>
41 *     SOAPConnectionFactory factory = SOAPConnectionFactory.newInstance();
42 *     SOAPConnection con = factory.createConnection();
43 * </PRE>
44 * A <code>SOAPConnection</code> object can be used to send messages
45 * directly to a URL following the request/response paradigm. That is,
46 * messages are sent using the method <code>call</code>, which sends the
47 * message and then waits until it gets a reply.
48 */
```

```

49 public abstract class SOAPConnection {
50
51     /**
52      * Sends the given message to the specified endpoint and blocks until
53      * it has returned the response.
54      *
55      * @param request the <code>SOAPMessage</code> object to be sent
56      * @param to an <code>Object</code> that identifies
57      *           where the message should be sent. It is required to
58      *           support Objects of type
59      *           <code>java.lang.String</code>,
60      *           <code>java.net.URL</code>, and when JAXM is present
61      *           <code>javax.xml.messaging.URLEndpoint</code>
62      *
63      * @return the <code>SOAPMessage</code> object that is the response to the
64      *         message that was sent
65      * @throws SOAPException if there is a SOAP error
66      */
67     public abstract SOAPMessage call(SOAPMessage request,
68                                     Object to) throws SOAPException;
69
70     /**
71      * Gets a message from a specific endpoint and blocks until it receives,
72      *
73      * @param to an <code>Object</code> that identifies where
74      *           the request should be sent. Objects of type
75      *           <code>java.lang.String</code> and
76      *           <code>java.net.URL</code> must be supported.
77      *
78      * @return the <code>SOAPMessage</code> object that is the response to the
79      *         get message request
80      * @throws SOAPException if there is a SOAP error
81      * @since SAAJ 1.3
82      */
83     public SOAPMessage get(Object to)
84         throws SOAPException {
85         throw new UnsupportedOperationException("All subclasses of SOAPConnection must override get");
86     }
87
88     /**
89      * Closes this <code>SOAPConnection</code> object.
90      *
91      * @throws SOAPException if there is a SOAP error
92      */
93     public abstract void close()
94         throws SOAPException;
95 }

```

## 17.15 SOAPConnectionFactory.java

Secuencia 176: javax/xml/soap/SOAPConnectionFactory.java

```

1  /*

```

```
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 /**
29  * A factory for creating SOAPConnection objects. Implementation of this class
30  * is optional. If SOAPConnectionFactory.newInstance() throws an
31  * UnsupportedOperationException then the implementation does not support the
32  * SAAJ communication infrastructure. Otherwise SOAPConnection objects
33  * can be created by calling createConnection() on the newly
34  * created SOAPConnectionFactory object.
35  */
36 public abstract class SOAPConnectionFactory {
37     /**
38      * A constant representing the default value for a SOAPConnection
39      * object. The default is the point-to-point SOAP connection.
40      */
41     static final String DEFAULT_SOAP_CONNECTION_FACTORY
42         = "com.sun.xml.internal.messaging.saaj.client.p2p.HttpSOAPConnectionFactory";
43
44     /**
45      * A constant representing the SOAPConnection class.
46      */
47     static private final String SF_PROPERTY
48         = "javax.xml.soap.SOAPConnectionFactory";
49
50     /**
51      * Creates an instance of the default
52      * SOAPConnectionFactory object.
53      *
54      * @return a new instance of a default
```

```

55      *      <code>SOAPConnectionFactory</code> object
56      *
57      * @exception SOAPException if there was an error creating the
58      *      <code>SOAPConnectionFactory</code>
59      *
60      * @exception UnsupportedOperationException if newInstance is not
61      * supported.
62      */
63      public static SOAPConnectionFactory newInstance()
64          throws SOAPException, UnsupportedOperationException
65      {
66          try {
67              return (SOAPConnectionFactory)
68                  FactoryFinder.find(SF_PROPERTY,
69                      DEFAULT_SOAP_CONNECTION_FACTORY);
70          } catch (Exception ex) {
71              throw new SOAPException("Unable to create SOAP connection factory: "
72                  + ex.getMessage());
73          }
74      }
75
76      /**
77       * Create a new <code>SOAPConnection</code>.
78       *
79       * @return the new <code>SOAPConnection</code> object.
80       *
81       * @exception SOAPException if there was an exception creating the
82       * <code>SOAPConnection</code> object.
83       */
84      public abstract SOAPConnection createConnection()
85          throws SOAPException;
86  }

```

## 17.16 SOAPConstants.java

Secuencia 177: javax/xml/soap/SOAPConstants.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.soap;
27
28 import javax.xml.namespace.QName;
29
30 /**
31  * The definition of constants pertaining to the SOAP protocol.
32  */
33 public interface SOAPConstants {
34     /**
35      * Used to create MessageFactory instances that create
36      * SOAPMessages whose concrete type is based on the
37      * Content-Type MIME header passed to the
38      * createMessage method. If no Content-Type
39      * header is passed then the createMessage may throw an
40      * IllegalArgumentException or, in the case of the no
41      * argument version of createMessage, an
42      * UnsupportedOperationException.
43      *
44      * @since SAAJ 1.3
45      */
46     public static final String DYNAMIC_SOAP_PROTOCOL = "Dynamic Protocol";
47
48     /**
49      * Used to create MessageFactory instances that create
50      * SOAPMessages whose behavior supports the SOAP 1.1 specification.
51      *
52      * @since SAAJ 1.3
53      */
54     public static final String SOAP_1_1_PROTOCOL = "SOAP 1.1 Protocol";
55
56     /**
57      * Used to create MessageFactory instances that create
58      * SOAPMessages whose behavior supports the SOAP 1.2
59      * specification
60      *
61      * @since SAAJ 1.3
62      */
63     public static final String SOAP_1_2_PROTOCOL = "SOAP 1.2 Protocol";
64
65     /**
66      * The default protocol: SOAP 1.1 for backwards compatibility.
67      *
68      * @since SAAJ 1.3
69      */
70     public static final String DEFAULT_SOAP_PROTOCOL = SOAP_1_1_PROTOCOL;
```



```
71
72 /**
73  * The namespace identifier for the SOAP 1.1 envelope.
74  * @since SAAJ 1.3
75  */
76 public static final String
77     URI_NS_SOAP_1_1_ENVELOPE = "http://schemas.xmlsoap.org/soap/envelope/";
78 /**
79  * The namespace identifier for the SOAP 1.2 envelope.
80  * @since SAAJ 1.3
81  */
82 public static final String
83     URI_NS_SOAP_1_2_ENVELOPE = "http://www.w3.org/2003/05/soap-envelope";
84
85 /**
86  * The namespace identifier for the SOAP 1.1 envelope, All SOAPElements in this
87  * namespace are defined by the SOAP 1.1 specification.
88  */
89 public static final String
90     URI_NS_SOAP_ENVELOPE = URI_NS_SOAP_1_1_ENVELOPE;
91
92 /**
93  * The namespace identifier for the SOAP 1.1 encoding.
94  * An attribute named <code>encodingStyle</code> in the
95  * <code>URI_NS_SOAP_ENVELOPE</code> namespace and set to the value
96  * <code>URI_NS_SOAP_ENCODING</code> can be added to an element to indicate
97  * that it is encoded using the rules in section 5 of the SOAP 1.1
98  * specification.
99  */
100 public static final String
101     URI_NS_SOAP_ENCODING = "http://schemas.xmlsoap.org/soap/encoding/";
102
103 /**
104  * The namespace identifier for the SOAP 1.2 encoding.
105  * @since SAAJ 1.3
106  */
107 public static final String
108     URI_NS_SOAP_1_2_ENCODING = "http://www.w3.org/2003/05/soap-encoding";
109
110 /**
111  * The media type of the <code>Content-Type</code> MIME header in SOAP 1.1.
112  * @since SAAJ 1.3
113  */
114 public static final String
115     SOAP_1_1_CONTENT_TYPE = "text/xml";
116
117 /**
118  * The media type of the <code>Content-Type</code> MIME header in SOAP 1.2.
119  * @since SAAJ 1.3
120  */
121 public static final String
122     SOAP_1_2_CONTENT_TYPE = "application/soap+xml";
123
```

```
124 /**
125  * The URI identifying the next application processing a SOAP request as the intended
126  * actor for a SOAP 1.1 header entry (see section 4.2.2 of the SOAP 1.1 specification).
127  * <p>
128  * This value can be passed to
129  * {@link SOAPHeader#examineMustUnderstandHeaderElements(String)},
130  * {@link SOAPHeader#examineHeaderElements(String)} and
131  * {@link SOAPHeader#extractHeaderElements(String)}
132  */
133 public static final String
134     URI_SOAP_ACTOR_NEXT = "http://schemas.xmlsoap.org/soap/actor/next";
135
136 /**
137  * The URI identifying the next application processing a SOAP request as the intended
138  * role for a SOAP 1.2 header entry (see section 2.2 of part 1 of the SOAP 1.2
139  * specification).
140  * @since SAAJ 1.3
141  */
142 public static final String
143     URI_SOAP_1_2_ROLE_NEXT = URI_NS_SOAP_1_2_ENVELOPE + "/role/next";
144
145 /**
146  * The URI specifying the role None in SOAP 1.2.
147  * @since SAAJ 1.3
148  */
149 public static final String
150     URI_SOAP_1_2_ROLE_NONE = URI_NS_SOAP_1_2_ENVELOPE + "/role/none";
151
152 /**
153  * The URI identifying the ultimate receiver of the SOAP 1.2 message.
154  * @since SAAJ 1.3
155  */
156 public static final String
157     URI_SOAP_1_2_ROLE_ULTIMATE_RECEIVER =
158     URI_NS_SOAP_1_2_ENVELOPE + "/role/ultimateReceiver";
159
160 /**
161  * The default namespace prefix for http://www.w3.org/2003/05/soap-envelope
162  * @since SAAJ 1.3
163  */
164 public static final String SOAP_ENV_PREFIX = "env";
165
166 /**
167  * SOAP 1.2 VersionMismatch Fault
168  * @since SAAJ 1.3
169  */
170 public static final QName SOAP_VERSIONMISMATCH_FAULT =
171     new QName(URI_NS_SOAP_1_2_ENVELOPE, "VersionMismatch", SOAP_ENV_PREFIX);
172
173 /**
174  * SOAP 1.2 MustUnderstand Fault
175  * @since SAAJ 1.3
176  */
```

```

177     public static final QName SOAP_MUSTUNDERSTAND_FAULT =
178         new QName(URI_NS_SOAP_1_2_ENVELOPE, "MustUnderstand", SOAP_ENV_PREFIX);
179
180     /**
181      * SOAP 1.2 DataEncodingUnknown Fault
182      * @since SAAJ 1.3
183      */
184     public static final QName SOAP_DATAENCODINGUNKNOWN_FAULT =
185         new QName(URI_NS_SOAP_1_2_ENVELOPE, "DataEncodingUnknown", SOAP_ENV_PREFIX);
186
187     /**
188      * SOAP 1.2 Sender Fault
189      * @since SAAJ 1.3
190      */
191     public static final QName SOAP_SENDER_FAULT =
192         new QName(URI_NS_SOAP_1_2_ENVELOPE, "Sender", SOAP_ENV_PREFIX);
193
194     /**
195      * SOAP 1.2 Receiver Fault
196      * @since SAAJ 1.3
197      */
198     public static final QName SOAP_RECEIVER_FAULT =
199         new QName(URI_NS_SOAP_1_2_ENVELOPE, "Receiver", SOAP_ENV_PREFIX);
200
201 }

```

## 17.17 SOAPElement.java

Secuencia 178: javax/xml/soap/SOAPElement.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */

```

```

25
26 package javax.xml.soap;
27
28 import java.util.Iterator;
29
30 import javax.xml.namespace.QName;
31
32 /**
33  * An object representing an element of a SOAP message that is allowed but not
34  * specifically prescribed by a SOAP specification. This interface serves as the
35  * base interface for those objects that are specifically prescribed by a SOAP
36  * specification.
37  * <p>
38  * Methods in this interface that are required to return SAAJ specific objects
39  * may "silently" replace nodes in the tree as required to successfully return
40  * objects of the correct type. See {@link #getChildElements()} and
41  * {@link <a href="package-summary.html#package_description">javax.xml.soap<a>}
42  * for details.
43  */
44 public interface SOAPElement extends Node, org.w3c.dom.Element {
45
46     /**
47      * Creates a new <code>SOAPElement</code> object initialized with the
48      * given <code>Name</code> object and adds the new element to this
49      * <code>SOAPElement</code> object.
50      * <P>
51      * This method may be deprecated in a future release of SAAJ in favor of
52      * addChildElement(javax.xml.namespace.QName)
53      *
54      * @param name a <code>Name</code> object with the XML name for the
55      *             new element
56      *
57      * @return the new <code>SOAPElement</code> object that was created
58      * @exception SOAPException if there is an error in creating the
59      *             <code>SOAPElement</code> object
60      * @see SOAPElement#addChildElement(javax.xml.namespace.QName)
61      */
62     public SOAPElement addChildElement(Name name) throws SOAPException;
63
64     /**
65      * Creates a new <code>SOAPElement</code> object initialized with the given
66      * <code>QName</code> object and adds the new element to this <code>SOAPElement</code>
67      * object. The <i>namespace</i>, <i>localname</i> and <i>prefix</i> of the new
68      * <code>SOAPElement</code> are all taken from the <code>qname</code> argument.
69      *
70      * @param qname a <code>QName</code> object with the XML name for the
71      *             new element
72      *
73      * @return the new <code>SOAPElement</code> object that was created
74      * @exception SOAPException if there is an error in creating the
75      *             <code>SOAPElement</code> object
76      * @see SOAPElement#addChildElement(Name)
77      * @since SAAJ 1.3

```

```
78     */
79     public SOAPElement addChildElement(QName qname) throws SOAPException;
80
81     /**
82     * Creates a new <code>SOAPElement</code> object initialized with the
83     * specified local name and adds the new element to this
84     * <code>SOAPElement</code> object.
85     * The new <code>SOAPElement</code> inherits any in-scope default namespace.
86     *
87     * @param localName a <code>String</code> giving the local name for
88     *     the element
89     * @return the new <code>SOAPElement</code> object that was created
90     * @exception SOAPException if there is an error in creating the
91     *     <code>SOAPElement</code> object
92     */
93     public SOAPElement addChildElement(String localName) throws SOAPException;
94
95     /**
96     * Creates a new <code>SOAPElement</code> object initialized with the
97     * specified local name and prefix and adds the new element to this
98     * <code>SOAPElement</code> object.
99     *
100    * @param localName a <code>String</code> giving the local name for
101    *     the new element
102    * @param prefix a <code>String</code> giving the namespace prefix for
103    *     the new element
104    *
105    * @return the new <code>SOAPElement</code> object that was created
106    * @exception SOAPException if the <code>prefix</code> is not valid in the
107    *     context of this <code>SOAPElement</code> or if there is an error in creating the
108    *     <code>SOAPElement</code> object
109    */
110    public SOAPElement addChildElement(String localName, String prefix)
111        throws SOAPException;
112
113    /**
114    * Creates a new <code>SOAPElement</code> object initialized with the
115    * specified local name, prefix, and URI and adds the new element to this
116    * <code>SOAPElement</code> object.
117    *
118    * @param localName a <code>String</code> giving the local name for
119    *     the new element
120    * @param prefix a <code>String</code> giving the namespace prefix for
121    *     the new element
122    * @param uri a <code>String</code> giving the URI of the namespace
123    *     to which the new element belongs
124    *
125    * @return the new <code>SOAPElement</code> object that was created
126    * @exception SOAPException if there is an error in creating the
127    *     <code>SOAPElement</code> object
128    */
129    public SOAPElement addChildElement(String localName, String prefix,
130        String uri)
```

```

131         throws SOAPException;
132
133     /**
134     * Add a <code>SOAPElement</code> as a child of this
135     * <code>SOAPElement</code> instance. The <code>SOAPElement</code>
136     * is expected to be created by a
137     * <code>SOAPFactory</code>. Callers should not rely on the
138     * element instance being added as is into the XML
139     * tree. Implementations could end up copying the content
140     * of the <code>SOAPElement</code> passed into an instance of
141     * a different <code>SOAPElement</code> implementation. For
142     * instance if <code>addChildElement()</code> is called on a
143     * <code>SOAPHeader</code>, <code>element</code> will be copied
144     * into an instance of a <code>SOAPHeaderElement</code>.
145     *
146     * <P>The fragment rooted in <code>element</code> is either added
147     * as a whole or not at all, if there was an error.
148     *
149     * <P>The fragment rooted in <code>element</code> cannot contain
150     * elements named "Envelope", "Header" or "Body" and in the SOAP
151     * namespace. Any namespace prefixes present in the fragment
152     * should be fully resolved using appropriate namespace
153     * declarations within the fragment itself.
154     *
155     * @param element the <code>SOAPElement</code> to be added as a
156     *             new child
157     *
158     * @exception SOAPException if there was an error in adding this
159     *             element as a child
160     *
161     * @return an instance representing the new SOAP element that was
162     *             actually added to the tree.
163     */
164     public SOAPElement addChildElement(SOAPElement element)
165         throws SOAPException;
166
167     /**
168     * Detaches all children of this <code>SOAPElement</code>.
169     * <p>
170     * This method is useful for rolling back the construction of partially
171     * completed <code>SOAPHeaders</code> and <code>SOAPBodys</code> in
172     * preparation for sending a fault when an error condition is detected. It
173     * is also useful for recycling portions of a document within a SOAP
174     * message.
175     *
176     * @since SAAJ 1.2
177     */
178     public abstract void removeContents();
179
180     /**
181     * Creates a new <code>Text</code> object initialized with the given
182     * <code>String</code> and adds it to this <code>SOAPElement</code> object.
183     *

```

```
184 * @param text a <code>String</code> object with the textual content to be added
185 *
186 * @return the <code>SOAPElement</code> object into which
187 *         the new <code>Text</code> object was inserted
188 * @exception SOAPException if there is an error in creating the
189 *         new <code>Text</code> object or if it is not legal to
190 *         attach it as a child to this
191 *         <code>SOAPElement</code>
192 */
193 public SOAPElement addTextNode(String text) throws SOAPException;
194
195 /**
196 * Adds an attribute with the specified name and value to this
197 * <code>SOAPElement</code> object.
198 *
199 * @param name a <code>Name</code> object with the name of the attribute
200 * @param value a <code>String</code> giving the value of the attribute
201 * @return the <code>SOAPElement</code> object into which the attribute was
202 *         inserted
203 *
204 * @exception SOAPException if there is an error in creating the
205 *         Attribute, or it is invalid to set
206 *         an attribute with <code>Name</code>
207 *         <code>name</code> on this SOAPElement.
208 * @see SOAPElement#addAttribute(javax.xml.namespace.QName, String)
209 */
210 public SOAPElement addAttribute(Name name, String value)
211     throws SOAPException;
212
213 /**
214 * Adds an attribute with the specified name and value to this
215 * <code>SOAPElement</code> object.
216 *
217 * @param qname a <code>QName</code> object with the name of the attribute
218 * @param value a <code>String</code> giving the value of the attribute
219 * @return the <code>SOAPElement</code> object into which the attribute was
220 *         inserted
221 *
222 * @exception SOAPException if there is an error in creating the
223 *         Attribute, or it is invalid to set
224 *         an attribute with <code>QName</code>
225 *         <code>qname</code> on this SOAPElement.
226 * @see SOAPElement#addAttribute(Name, String)
227 * @since SAAJ 1.3
228 */
229 public SOAPElement addAttribute(QName qname, String value)
230     throws SOAPException;
231
232 /**
233 * Adds a namespace declaration with the specified prefix and URI to this
234 * <code>SOAPElement</code> object.
235 *
236 * @param prefix a <code>String</code> giving the prefix of the namespace
```

```
237 * @param uri a <code>String</code> giving the uri of the namespace
238 * @return the <code>SOAPElement</code> object into which this
239 *         namespace declaration was inserted.
240 *
241 * @exception SOAPException if there is an error in creating the
242 *         namespace
243 */
244 public SOAPElement addNamespaceDeclaration(String prefix, String uri)
245     throws SOAPException;
246
247 /**
248 * Returns the value of the attribute with the specified name.
249 *
250 * @param name a <code>Name</code> object with the name of the attribute
251 * @return a <code>String</code> giving the value of the specified
252 *         attribute, Null if there is no such attribute
253 * @see SOAPElement#getAttributeValue(javax.xml.namespace.QName)
254 */
255 public String getAttributeValue(Name name);
256
257 /**
258 * Returns the value of the attribute with the specified qname.
259 *
260 * @param qname a <code>QName</code> object with the qname of the attribute
261 * @return a <code>String</code> giving the value of the specified
262 *         attribute, Null if there is no such attribute
263 * @see SOAPElement#getAttributeValue(Name)
264 * @since SAAJ 1.3
265 */
266 public String getAttributeValue(QName qname);
267
268 /**
269 * Returns an <code>Iterator</code> over all of the attribute
270 * <code>Name</code> objects in this
271 * <code>SOAPElement</code> object. The iterator can be used to get
272 * the attribute names, which can then be passed to the method
273 * <code>getAttributeValue</code> to retrieve the value of each
274 * attribute.
275 *
276 * @see SOAPElement#getAllAttributesAsQNames()
277 * @return an iterator over the names of the attributes
278 */
279 public Iterator getAllAttributes();
280
281 /**
282 * Returns an <code>Iterator</code> over all of the attributes
283 * in this <code>SOAPElement</code> as <code>QName</code> objects.
284 * The iterator can be used to get the attribute QName, which can then
285 * be passed to the method <code>getAttributeValue</code> to retrieve
286 * the value of each attribute.
287 *
288 * @return an iterator over the QNames of the attributes
289 * @see SOAPElement#getAllAttributes()
```



```
290     * @since SAAJ 1.3
291     */
292     public Iterator getAllAttributesAsQNames();
293
294
295     /**
296     * Returns the URI of the namespace that has the given prefix.
297     *
298     * @param prefix a <code>String</code> giving the prefix of the namespace
299     *             for which to search
300     * @return a <code>String</code> with the uri of the namespace that has
301     *         the given prefix
302     */
303     public String getNamespaceURI(String prefix);
304
305     /**
306     * Returns an <code>Iterator</code> over the namespace prefix
307     * <code>String</code>s declared by this element. The prefixes returned by
308     * this iterator can be passed to the method
309     * <code>getNamespaceURI</code> to retrieve the URI of each namespace.
310     *
311     * @return an iterator over the namespace prefixes in this
312     *         <code>SOAPElement</code> object
313     */
314     public Iterator getNamespacePrefixes();
315
316     /**
317     * Returns an <code>Iterator</code> over the namespace prefix
318     * <code>String</code>s visible to this element. The prefixes returned by
319     * this iterator can be passed to the method
320     * <code>getNamespaceURI</code> to retrieve the URI of each namespace.
321     *
322     * @return an iterator over the namespace prefixes are within scope of this
323     *         <code>SOAPElement</code> object
324     *
325     * @since SAAJ 1.2
326     */
327     public Iterator getVisibleNamespacePrefixes();
328
329     /**
330     * Creates a <code>QName</code> whose namespace URI is the one associated
331     * with the parameter, <code>prefix</code>, in the context of this
332     * <code>SOAPElement</code>. The remaining elements of the new
333     * <code>QName</code> are taken directly from the parameters,
334     * <code>localName</code> and <code>prefix</code>.
335     *
336     * @param localName
337     *             a <code>String</code> containing the local part of the name.
338     * @param prefix
339     *             a <code>String</code> containing the prefix for the name.
340     *
341     * @return a <code>QName</code> with the specified <code>localName</code>
342     *         and <code>prefix</code>, and with a namespace that is associated
```

```

343     *      with the <code>prefix</code> in the context of this
344     *      <code>SOAPElement</code>. This namespace will be the same as
345     *      the one that would be returned by
346     *      <code>{@link #getNamespaceURI(String)}</code> if it were given
347     *      <code>prefix</code> as it's parameter.
348     *
349     * @exception SOAPException if the <code>QName</code> cannot be created.
350     *
351     * @since SAAJ 1.3
352     */
353     public QName createQName(String localName, String prefix)
354         throws SOAPException;
355     /**
356     * Returns the name of this <code>SOAPElement</code> object.
357     *
358     * @return a <code>Name</code> object with the name of this
359     *         <code>SOAPElement</code> object
360     */
361     public Name getElementName();
362
363     /**
364     * Returns the qname of this <code>SOAPElement</code> object.
365     *
366     * @return a <code>QName</code> object with the qname of this
367     *         <code>SOAPElement</code> object
368     * @see SOAPElement#getElementName()
369     * @since SAAJ 1.3
370     */
371     public QName getElementQName();
372
373     /**
374     * Changes the name of this <code>Element</code> to <code>newName</code> if
375     * possible. SOAP Defined elements such as SOAPEnvelope, SOAPHeader, SOAPBody
376     * etc. cannot have their names changed using this method. Any attempt to do
377     * so will result in a SOAPException being thrown.
378     * <P>
379     * Callers should not rely on the element instance being renamed as is.
380     * Implementations could end up copying the content of the
381     * <code>SOAPElement</code> to a renamed instance.
382     *
383     * @param newName the new name for the <code>Element</code>.
384     *
385     * @exception SOAPException if changing the name of this <code>Element</code>
386     *         is not allowed.
387     * @return The renamed Node
388     *
389     * @since SAAJ 1.3
390     */
391     public SOAPElement setElementQName(QName newName) throws SOAPException;
392
393     /**
394     * Removes the attribute with the specified name.
395     *

```

```

396     * @param name the Name object with the name of the
397     *     attribute to be removed
398     * @return true if the attribute was
399     *     removed successfully; false if it was not
400     * @see SOAPElement#removeAttribute(javax.xml.namespace.QName)
401     */
402     public boolean removeAttribute(Name name);
403
404     /**
405     * Removes the attribute with the specified qname.
406     *
407     * @param qname the QName object with the qname of the
408     *     attribute to be removed
409     * @return true if the attribute was
410     *     removed successfully; false if it was not
411     * @see SOAPElement#removeAttribute(Name)
412     * @since SAAJ 1.3
413     */
414     public boolean removeAttribute(QName qname);
415
416     /**
417     * Removes the namespace declaration corresponding to the given prefix.
418     *
419     * @param prefix a String giving the prefix for which
420     *     to search
421     * @return true if the namespace declaration was
422     *     removed successfully; false if it was not
423     */
424     public boolean removeNamespaceDeclaration(String prefix);
425
426     /**
427     * Returns an Iterator over all the immediate child
428     * {@link Node}s of this element. This includes javax.xml.soap.Text
429     * objects as well as SOAPElement objects.
430     * 

431     * Calling this method may cause child Element,
432     * SOAPElement and org.w3c.dom.Text nodes to be
433     * replaced by SOAPElement, SOAPHeaderElement,
434     * SOAPBodyElement or javax.xml.soap.Text nodes as
435     * appropriate for the type of this parent node. As a result the calling
436     * application must treat any existing references to these child nodes that
437     * have been obtained through DOM APIs as invalid and either discard them or
438     * refresh them with the values returned by this Iterator. This
439     * behavior can be avoided by calling the equivalent DOM APIs. See
440     * {@link package-summary.html#package\_description} javax.xml.soap
441     * for more details.
442     *
443     * @return an iterator with the content of this SOAPElement
444     *     object
445     */
446     public Iterator getChildElements();
447
448     /**


```

```

449 * Returns an Iterator over all the immediate child
450 * {@link Node}s of this element with the specified name. All of these
451 * children will be SOAPElement nodes.
452 * <p>
453 * Calling this method may cause child Element,
454 * SOAPElement and org.w3c.dom.Text nodes to be
455 * replaced by SOAPElement, SOAPHeaderElement,
456 * SOAPBodyElement or javax.xml.soap.Text nodes as
457 * appropriate for the type of this parent node. As a result the calling
458 * application must treat any existing references to these child nodes that
459 * have been obtained through DOM APIs as invalid and either discard them or
460 * refresh them with the values returned by this Iterator. This
461 * behavior can be avoided by calling the equivalent DOM APIs. See
462 * {@link <a HREF="package-summary.html#package_description">javax.xml.soap<a>}
463 * for more details.
464 *
465 * @param name a Name object with the name of the child
466 *         elements to be returned
467 *
468 * @return an Iterator object over all the elements
469 *         in this SOAPElement object with the
470 *         specified name
471 * @see SOAPElement#getChildElements(javax.xml.namespace.QName)
472 */
473 public Iterator getChildElements(Name name);
474
475 /**
476 * Returns an Iterator over all the immediate child
477 * {@link Node}s of this element with the specified QName. All of these
478 * children will be SOAPElement nodes.
479 * <p>
480 * Calling this method may cause child Element,
481 * SOAPElement and org.w3c.dom.Text nodes to be
482 * replaced by SOAPElement, SOAPHeaderElement,
483 * SOAPBodyElement or javax.xml.soap.Text nodes as
484 * appropriate for the type of this parent node. As a result the calling
485 * application must treat any existing references to these child nodes that
486 * have been obtained through DOM APIs as invalid and either discard them or
487 * refresh them with the values returned by this Iterator. This
488 * behavior can be avoided by calling the equivalent DOM APIs. See
489 * {@link <a HREF="package-summary.html#package_description">javax.xml.soap<a>}
490 * for more details.
491 *
492 * @param QName a QName object with the QName of the child
493 *         elements to be returned
494 *
495 * @return an Iterator object over all the elements
496 *         in this SOAPElement object with the
497 *         specified QName
498 * @see SOAPElement#getChildElements(Name)
499 * @since SAAJ 1.3
500 */
501 public Iterator getChildElements(QName QName);

```

```

502
503 /**
504  * Sets the encoding style for this <code>SOAPElement</code> object
505  * to one specified.
506  *
507  * @param encodingStyle a <code>String</code> giving the encoding style
508  *
509  * @exception IllegalArgumentException if there was a problem in the
510  *         encoding style being set.
511  * @exception SOAPException if setting the encodingStyle is invalid for this SOAPElement.
512  * @see #getEncodingStyle
513  */
514 public void setEncodingStyle(String encodingStyle)
515     throws SOAPException;
516 /**
517  * Returns the encoding style for this <code>SOAPElement</code> object.
518  *
519  * @return a <code>String</code> giving the encoding style
520  *
521  * @see #setEncodingStyle
522  */
523 public String getEncodingStyle();
524 }

```

## 17.18 SOAPElementFactory.java

Secuencia 179: javax/xml/soap/SOAPElementFactory.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;

```

```

27
28 /**
29  * <code>SOAPElementFactory</code> is a factory for XML fragments that
30  * will eventually end up in the SOAP part. These fragments
31  * can be inserted as children of the <code>SOAPHeader</code> or
32  * <code>SOAPBody</code> or <code>SOAPEnvelope</code>.
33  *
34  * <p>Elements created using this factory do not have the properties
35  * of an element that lives inside a SOAP header document. These
36  * elements are copied into the XML document tree when they are
37  * inserted.
38  * @deprecated - Use <code>javax.xml.soap.SOAPFactory</code> for creating SOAPElements.
39  * @see javax.xml.soap.SOAPFactory
40  */
41 public class SOAPElementFactory {
42
43     private SOAPFactory soapFactory;
44
45     private SOAPElementFactory(SOAPFactory soapFactory) {
46         this.soapFactory = soapFactory;
47     }
48
49     /**
50      * Create a <code>SOAPElement</code> object initialized with the
51      * given <code>Name</code> object.
52      *
53      * @param name a <code>Name</code> object with the XML name for
54      *             the new element
55      *
56      * @return the new <code>SOAPElement</code> object that was
57      *         created
58      *
59      * @exception SOAPException if there is an error in creating the
60      *         <code>SOAPElement</code> object
61      *
62      * @deprecated Use
63      *     javax.xml.soap.SOAPFactory.createElement(javax.xml.soap.Name)
64      *     instead
65      *
66      * @see javax.xml.soap.SOAPFactory#createElement(javax.xml.soap.Name)
67      * @see javax.xml.soap.SOAPFactory#createElement(javax.xml.namespace.QName)
68      */
69     public SOAPElement create(Name name) throws SOAPException {
70         return soapFactory.createElement(name);
71     }
72
73     /**
74      * Create a <code>SOAPElement</code> object initialized with the
75      * given local name.
76      *
77      * @param localName a <code>String</code> giving the local name for
78      *                  the new element
79      *

```

```

80     * @return the new <code>SOAPElement</code> object that was
81     *         created
82     *
83     * @exception SOAPException if there is an error in creating the
84     *         <code>SOAPElement</code> object
85     *
86     * @deprecated Use
87     *         javax.xml.soap.SOAPFactory.createElement(String localName) instead
88     *
89     * @see javax.xml.soap.SOAPFactory#createElement(java.lang.String)
90     */
91     public SOAPElement create(String localName) throws SOAPException {
92         return soapFactory.createElement(localName);
93     }
94
95     /**
96     * Create a new <code>SOAPElement</code> object with the given
97     * local name, prefix and uri.
98     *
99     * @param localName a <code>String</code> giving the local name
100    *                  for the new element
101    * @param prefix the prefix for this <code>SOAPElement</code>
102    * @param uri a <code>String</code> giving the URI of the
103    *            namespace to which the new element belongs
104    *
105    * @exception SOAPException if there is an error in creating the
106    *            <code>SOAPElement</code> object
107    *
108    * @deprecated Use
109    *         javax.xml.soap.SOAPFactory.createElement(String localName,
110    *             String prefix,
111    *             String uri)
112    *         instead
113    *
114    * @see javax.xml.soap.SOAPFactory#createElement(java.lang.String, java.lang.String, java.lang.
115    *         String)
116    */
117    public SOAPElement create(String localName, String prefix, String uri)
118        throws SOAPException {
119        return soapFactory.createElement(localName, prefix, uri);
120    }
121
122    /**
123     * Creates a new instance of <code>SOAPElementFactory</code>.
124     *
125     * @return a new instance of a <code>SOAPElementFactory</code>
126     *
127     * @exception SOAPException if there was an error creating the
128     *            default <code>SOAPElementFactory</code>
129     */
130    public static SOAPElementFactory newInstance() throws SOAPException {
131        try {
132            return new SOAPElementFactory(SOAPFactory.newInstance());
133        }

```

```

132     } catch (Exception ex) {
133         throw new SOAPException(
134             "Unable to create SOAP Element Factory: " + ex.getMessage());
135     }
136 }
137 }

```

## 17.19 SOAPEnvelope.java

Secuencia 180: javax/xml/soap/SOAPEnvelope.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28
29 /**
30 * The container for the SOAPHeader and SOAPBody portions of a
31 * <code>SOAPPart</code> object. By default, a <code>SOAPMessage</code>
32 * object is created with a <code>SOAPPart</code> object that has a
33 * <code>SOAPEnvelope</code> object. The <code>SOAPEnvelope</code> object
34 * by default has an empty <code>SOAPBody</code> object and an empty
35 * <code>SOAPHeader</code> object. The <code>SOAPBody</code> object is
36 * required, and the <code>SOAPHeader</code> object, though
37 * optional, is used in the majority of cases. If the
38 * <code>SOAPHeader</code> object is not needed, it can be deleted,
39 * which is shown later.
40 * <P>
41 * A client can access the <code>SOAPHeader</code> and <code>SOAPBody</code>
42 * objects by calling the methods <code>SOAPEnvelope.getHeader</code> and
43 * <code>SOAPEnvelope.getBody</code>. The

```



```

44 * following lines of code use these two methods after starting with
45 * the <code>SOAPMessage</code>
46 * object <i>message</i> to get the <code>SOAPPart</code> object <i>sp</i>,
47 * which is then used to get the <code>SOAPEnvelope</code> object <i>se</i>.
48 *
49 * <PRE>
50 *     SOAPPart sp = message.getSOAPPart();
51 *     SOAPEnvelope se = sp.getEnvelope();
52 *     SOAPHeader sh = se.getHeader();
53 *     SOAPBody sb = se.getBody();
54 * </PRE>
55 * <P>
56 * It is possible to change the body or header of a <code>SOAPEnvelope</code>
57 * object by retrieving the current one, deleting it, and then adding
58 * a new body or header. The <code>javax.xml.soap.Node</code> method
59 * <code>deleteNode</code> deletes the XML element (node) on which it is
60 * called. For example, the following line of code deletes the
61 * <code>SOAPBody</code> object that is retrieved by the method <code>getBody</code>.
62 * <PRE>
63 *     se.getBody().detachNode();
64 * </PRE>
65 * To create a <code>SOAPHeader</code> object to replace the one that was removed,
66 * a client uses
67 * the method <code>SOAPEnvelope.addHeader</code>, which creates a new header and
68 * adds it to the <code>SOAPEnvelope</code> object. Similarly, the method
69 * <code>addBody</code> creates a new <code>SOAPBody</code> object and adds
70 * it to the <code>SOAPEnvelope</code> object. The following code fragment
71 * retrieves the current header, removes it, and adds a new one. Then
72 * it retrieves the current body, removes it, and adds a new one.
73 *
74 * <PRE>
75 *     SOAPPart sp = message.getSOAPPart();
76 *     SOAPEnvelope se = sp.getEnvelope();
77 *     se.getHeader().detachNode();
78 *     SOAPHeader sh = se.addHeader();
79 *     se.getBody().detachNode();
80 *     SOAPBody sb = se.addBody();
81 * </PRE>
82 * It is an error to add a <code>SOAPBody</code> or <code>SOAPHeader</code>
83 * object if one already exists.
84 * <P>
85 * The <code>SOAPEnvelope</code> interface provides three methods for creating
86 * <code>Name</code> objects. One method creates <code>Name</code> objects with
87 * a local name, a namespace prefix, and a namespace URI. The second method creates
88 * <code>Name</code> objects with a local name and a namespace prefix, and the third
89 * creates <code>Name</code> objects with just a local name. The following line of
90 * code, in which <i>se</i> is a <code>SOAPEnvelope</code> object, creates a new
91 * <code>Name</code> object with all three.
92 * <PRE>
93 *     Name name = se.createName("GetLastTradePrice", "WOMBAT",
94 *                               "http://www.wombat.org/trader");
95 * </PRE>
96 */

```

```

97 public interface SOAPEnvelope extends SOAPElement {
98
99     /**
100      * Creates a new <code>Name</code> object initialized with the
101      * given local name, namespace prefix, and namespace URI.
102      * <P>
103      * This factory method creates <code>Name</code> objects for use in
104      * the SOAP/XML document.
105      *
106      * @param localName a <code>String</code> giving the local name
107      * @param prefix a <code>String</code> giving the prefix of the namespace
108      * @param uri a <code>String</code> giving the URI of the namespace
109      * @return a <code>Name</code> object initialized with the given
110      *         local name, namespace prefix, and namespace URI
111      * @throws SOAPException if there is a SOAP error
112      */
113     public abstract Name createName(String localName, String prefix,
114                                     String uri)
115         throws SOAPException;
116
117     /**
118      * Creates a new <code>Name</code> object initialized with the
119      * given local name.
120      * <P>
121      * This factory method creates <code>Name</code> objects for use in
122      * the SOAP/XML document.
123      *
124      * @param localName a <code>String</code> giving the local name
125      * @return a <code>Name</code> object initialized with the given
126      *         local name
127      * @throws SOAPException if there is a SOAP error
128      */
129     public abstract Name createName(String localName)
130         throws SOAPException;
131
132     /**
133      * Returns the <code>SOAPHeader</code> object for
134      * this <code>SOAPEnvelope</code> object.
135      * <P>
136      * A new <code>SOAPMessage</code> object is by default created with a
137      * <code>SOAPEnvelope</code> object that contains an empty
138      * <code>SOAPHeader</code> object. As a result, the method
139      * <code>getHeader</code> will always return a <code>SOAPHeader</code>
140      * object unless the header has been removed and a new one has not
141      * been added.
142      *
143      * @return the <code>SOAPHeader</code> object or <code>null</code> if
144      *         there is none
145      * @exception SOAPException if there is a problem obtaining the
146      *         <code>SOAPHeader</code> object
147      */
148     public SOAPHeader getHeader() throws SOAPException;
149

```

```
150 /**
151  * Returns the <code>SOAPBody</code> object associated with this
152  * <code>SOAPEnvelope</code> object.
153  * <P>
154  * A new <code>SOAPMessage</code> object is by default created with a
155  * <code>SOAPEnvelope</code> object that contains an empty
156  * <code>SOAPBody</code> object. As a result, the method
157  * <code>getBody</code> will always return a <code>SOAPBody</code>
158  * object unless the body has been removed and a new one has not
159  * been added.
160  *
161  * @return the <code>SOAPBody</code> object for this
162  *         <code>SOAPEnvelope</code> object or <code>null</code>
163  *         if there is none
164  * @exception SOAPException if there is a problem obtaining the
165  *         <code>SOAPBody</code> object
166  */
167 public SOAPBody getBody() throws SOAPException;
168 /**
169  * Creates a <code>SOAPHeader</code> object and sets it as the
170  * <code>SOAPHeader</code> object for this <code>SOAPEnvelope</code>
171  * object.
172  * <P>
173  * It is illegal to add a header when the envelope already
174  * contains a header. Therefore, this method should be called
175  * only after the existing header has been removed.
176  *
177  * @return the new <code>SOAPHeader</code> object
178  *
179  * @exception SOAPException if this
180  *         <code>SOAPEnvelope</code> object already contains a
181  *         valid <code>SOAPHeader</code> object
182  */
183 public SOAPHeader addHeader() throws SOAPException;
184 /**
185  * Creates a <code>SOAPBody</code> object and sets it as the
186  * <code>SOAPBody</code> object for this <code>SOAPEnvelope</code>
187  * object.
188  * <P>
189  * It is illegal to add a body when the envelope already
190  * contains a body. Therefore, this method should be called
191  * only after the existing body has been removed.
192  *
193  * @return the new <code>SOAPBody</code> object
194  *
195  * @exception SOAPException if this
196  *         <code>SOAPEnvelope</code> object already contains a
197  *         valid <code>SOAPBody</code> object
198  */
199 public SOAPBody addBody() throws SOAPException;
200 }
```

**17.20 SOAPException.java**

Secuencia 181: javax/xml/soap/SOAPException.java

```
1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 /**
29  * An exception that signals that a SOAP exception has occurred. A
30  * <code>SOAPException</code> object may contain a <code>String</code>
31  * that gives the reason for the exception, an embedded
32  * <code>Throwable</code> object, or both. This class provides methods
33  * for retrieving reason messages and for retrieving the embedded
34  * <code>Throwable</code> object.
35  *
36  * <P> Typical reasons for throwing a <code>SOAPException</code>
37  * object are problems such as difficulty setting a header, not being
38  * able to send a message, and not being able to get a connection with
39  * the provider. Reasons for embedding a <code>Throwable</code>
40  * object include problems such as input/output errors or a parsing
41  * problem, such as an error in parsing a header.
42  */
43 public class SOAPException extends Exception {
44     private Throwable cause;
45
46     /**
47      * Constructs a <code>SOAPException</code> object with no
48      * reason or embedded <code>Throwable</code> object.
49      */
50     public SOAPException() {
```

```
51     super();
52     this.cause = null;
53 }
54
55 /**
56  * Constructs a <code>SOAPException</code> object with the given
57  * <code>String</code> as the reason for the exception being thrown.
58  *
59  * @param reason a description of what caused the exception
60  */
61 public SOAPException(String reason) {
62     super(reason);
63     this.cause = null;
64 }
65
66 /**
67  * Constructs a <code>SOAPException</code> object with the given
68  * <code>String</code> as the reason for the exception being thrown
69  * and the given <code>Throwable</code> object as an embedded
70  * exception.
71  *
72  * @param reason a description of what caused the exception
73  * @param cause a <code>Throwable</code> object that is to
74  *             be embedded in this <code>SOAPException</code> object
75  */
76 public SOAPException(String reason, Throwable cause) {
77     super(reason);
78     initCause(cause);
79 }
80
81 /**
82  * Constructs a <code>SOAPException</code> object initialized
83  * with the given <code>Throwable</code> object.
84  */
85 public SOAPException(Throwable cause) {
86     super(cause.toString());
87     initCause(cause);
88 }
89
90 /**
91  * Returns the detail message for this <code>SOAPException</code>
92  * object.
93  * <P>
94  * If there is an embedded <code>Throwable</code> object, and if the
95  * <code>SOAPException</code> object has no detail message of its
96  * own, this method will return the detail message from the embedded
97  * <code>Throwable</code> object.
98  *
99  * @return the error or warning message for this
100  *        <code>SOAPException</code> or, if it has none, the
101  *        message of the embedded <code>Throwable</code> object,
102  *        if there is one
103  */
```

```

104 public String getMessage() {
105     String message = super.getMessage();
106     if (message == null && cause != null) {
107         return cause.getMessage();
108     } else {
109         return message;
110     }
111 }
112
113 /**
114  * Returns the Throwable object embedded in this
115  * SOAPException if there is one. Otherwise, this method
116  * returns null.
117  *
118  * @return the embedded Throwable object or null
119  *         if there is none
120  */
121
122 public Throwable getCause() {
123     return cause;
124 }
125
126 /**
127  * Initializes the cause field of this SOAPException
128  * object with the given Throwable object.
129  * <P>
130  * This method can be called at most once. It is generally called from
131  * within the constructor or immediately after the constructor has
132  * returned a new SOAPException object.
133  * If this SOAPException object was created with the
134  * constructor {@link #SOAPException(Throwable)} or
135  * {@link #SOAPException(String,Throwable)}, meaning that its
136  * cause field already has a value, this method cannot be
137  * called even once.
138  *
139  * @param cause the Throwable object that caused this
140  *              SOAPException object to be thrown. The value of this
141  *              parameter is saved for later retrieval by the
142  *              {@link #getCause()} method. A null value is
143  *              permitted and indicates that the cause is nonexistent or
144  *              unknown.
145  * @return a reference to this SOAPException instance
146  * @throws IllegalArgumentException if cause is this
147  *              Throwable object. (A Throwable object
148  *              cannot be its own cause.)
149  * @throws IllegalStateException if the cause for this SOAPException object
150  *              has already been initialized
151  */
152 public synchronized Throwable initCause(Throwable cause) {
153     if (this.cause != null) {
154         throw new IllegalStateException("Can't override cause");
155     }
156     if (cause == this) {

```

```
157         throw new IllegalArgumentException("Self-causation not permitted");
158     }
159     this.cause = cause;
160
161     return this;
162 }
163 }
```

## 17.21 SOAPFactory.java

Secuencia 182: javax/xml/soap/SOAPFactory.java

```
1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import javax.xml.namespace.QName;
29
30 import org.w3c.dom.Element;
31
32 /**
33  * <code>SOAPFactory</code> is a factory for creating various objects
34  * that exist in the SOAP XML tree.
35
36  * <code>SOAPFactory</code> can be
37  * used to create XML fragments that will eventually end up in the
38  * SOAP part. These fragments can be inserted as children of the
39  * {@link SOAPHeaderElement} or {@link SOAPBodyElement} or
40  * {@link SOAPEnvelope} or other {@link SOAPElement} objects.
41  *
42  * <code>SOAPFactory</code> also has methods to create
```

```

43  * <code>javax.xml.soap.Detail</code> objects as well as
44  * <code>java.xml.soap.Name</code> objects.
45  *
46  */
47  public abstract class SOAPFactory {
48
49      /**
50       * A constant representing the property used to lookup the name of
51       * a <code>SOAPFactory</code> implementation class.
52       */
53      static private final String SOAP_FACTORY_PROPERTY =
54          "javax.xml.soap.SOAPFactory";
55
56      /**
57       * Class name of default <code>SOAPFactory</code> implementation.
58       */
59      static final String DEFAULT_SOAP_FACTORY
60          = "com.sun.xml.internal.messaging.saa.j.soap.ver1_1.SOAPFactory1_1Impl";
61
62      /**
63       * Creates a <code>SOAPElement</code> object from an existing DOM
64       * <code>Element</code>. If the DOM <code>Element</code> that is passed in
65       * as an argument is already a <code>SOAPElement</code> then this method
66       * must return it unmodified without any further work. Otherwise, a new
67       * <code>SOAPElement</code> is created and a deep copy is made of the
68       * <code>domElement</code> argument. The concrete type of the return value
69       * will depend on the name of the <code>domElement</code> argument. If any
70       * part of the tree rooted in <code>domElement</code> violates SOAP rules, a
71       * <code>SOAPException</code> will be thrown.
72       *
73       * @param domElement - the <code>Element</code> to be copied.
74       *
75       * @return a new <code>SOAPElement</code> that is a copy of <code>domElement</code>.
76       *
77       * @exception SOAPException if there is an error in creating the
78       *         <code>SOAPElement</code> object
79       *
80       * @since SAAJ 1.3
81       */
82      public SOAPElement createElement(Element domElement) throws SOAPException {
83          throw new UnsupportedOperationException("createElement(org.w3c.dom.Element) must be
            overridden by all subclasses of SOAPFactory.");
84      }
85
86      /**
87       * Creates a <code>SOAPElement</code> object initialized with the
88       * given <code>Name</code> object. The concrete type of the return value
89       * will depend on the name given to the new <code>SOAPElement</code>. For
90       * instance, a new <code>SOAPElement</code> with the name
91       * "{http://www.w3.org/2003/05/soap-envelope}Envelope" would cause a
92       * <code>SOAPEnvelope</code> that supports SOAP 1.2 behavior to be created.
93       *
94       * @param name a <code>Name</code> object with the XML name for

```



```

95      *          the new element
96      *
97      * @return the new <code>SOAPElement</code> object that was
98      *          created
99      *
100     * @exception SOAPException if there is an error in creating the
101     *          <code>SOAPElement</code> object
102     * @see SOAPFactory#createElement(javax.xml.namespace.QName)
103     */
104     public abstract SOAPElement createElement(Name name) throws SOAPException;
105
106     /**
107     * Creates a <code>SOAPElement</code> object initialized with the
108     * given <code>QName</code> object. The concrete type of the return value
109     * will depend on the name given to the new <code>SOAPElement</code>. For
110     * instance, a new <code>SOAPElement</code> with the name
111     * "{http://www.w3.org/2003/05/soap-envelope}Envelope" would cause a
112     * <code>SOAPEnvelope</code> that supports SOAP 1.2 behavior to be created.
113     *
114     * @param qname a <code>QName</code> object with the XML name for
115     *          the new element
116     *
117     * @return the new <code>SOAPElement</code> object that was
118     *          created
119     *
120     * @exception SOAPException if there is an error in creating the
121     *          <code>SOAPElement</code> object
122     * @see SOAPFactory#createElement(Name)
123     * @since SAAJ 1.3
124     */
125     public SOAPElement createElement(QName qname) throws SOAPException {
126         throw new UnsupportedOperationException("createElement(QName) must be overridden by all
127         subclasses of SOAPFactory.");
128     }
129
130     /**
131     * Creates a <code>SOAPElement</code> object initialized with the
132     * given local name.
133     *
134     * @param localName a <code>String</code> giving the local name for
135     *          the new element
136     *
137     * @return the new <code>SOAPElement</code> object that was
138     *          created
139     *
140     * @exception SOAPException if there is an error in creating the
141     *          <code>SOAPElement</code> object
142     */
143     public abstract SOAPElement createElement(String localName)
144         throws SOAPException;
145
146     /**

```

```

147 * Creates a new <code>SOAPElement</code> object with the given
148 * local name, prefix and uri. The concrete type of the return value
149 * will depend on the name given to the new <code>SOAPElement</code>. For
150 * instance, a new <code>SOAPElement</code> with the name
151 * "{http://www.w3.org/2003/05/soap-envelope}Envelope" would cause a
152 * <code>SOAPEnvelope</code> that supports SOAP 1.2 behavior to be created.
153 *
154 * @param localName a <code>String</code> giving the local name
155 *                  for the new element
156 * @param prefix the prefix for this <code>SOAPElement</code>
157 * @param uri a <code>String</code> giving the URI of the
158 *            namespace to which the new element belongs
159 *
160 * @exception SOAPException if there is an error in creating the
161 *            <code>SOAPElement</code> object
162 */
163 public abstract SOAPElement createElement(
164     String localName,
165     String prefix,
166     String uri)
167     throws SOAPException;
168
169 /**
170 * Creates a new <code>Detail</code> object which serves as a container
171 * for <code>DetailEntry</code> objects.
172 * <P>
173 * This factory method creates <code>Detail</code> objects for use in
174 * situations where it is not practical to use the <code>SOAPFault</code>
175 * abstraction.
176 *
177 * @return a <code>Detail</code> object
178 * @throws SOAPException if there is a SOAP error
179 * @throws UnsupportedOperationException if the protocol specified
180 *         for the SOAPFactory was <code>DYNAMIC_SOAP_PROTOCOL</code>
181 */
182 public abstract Detail createDetail() throws SOAPException;
183
184 /**
185 * Creates a new <code>SOAPFault</code> object initialized with the given <code>reasonText</code>
186 *    >
187 *    and <code>faultCode</code>
188 * @param reasonText the ReasonText/FaultString for the fault
189 * @param faultCode the FaultCode for the fault
190 * @return a <code>SOAPFault</code> object
191 * @throws SOAPException if there is a SOAP error
192 * @since SAAJ 1.3
193 */
194 public abstract SOAPFault createFault(String reasonText, QName faultCode) throws SOAPException;
195
196 /**
197 * Creates a new default <code>SOAPFault</code> object
198 * @return a <code>SOAPFault</code> object
199 * @throws SOAPException if there is a SOAP error

```

```
199     *@since SAAJ 1.3
200     */
201     public abstract SOAPFault createFault() throws SOAPException;
202
203     /**
204      * Creates a new Name object initialized with the
205      * given local name, namespace prefix, and namespace URI.
206      * <P>
207      * This factory method creates Name objects for use in
208      * situations where it is not practical to use the SOAPEnvelope
209      * abstraction.
210      *
211      * @param localName a String giving the local name
212      * @param prefix a String giving the prefix of the namespace
213      * @param uri a String giving the URI of the namespace
214      * @return a Name object initialized with the given
215      *         local name, namespace prefix, and namespace URI
216      * @throws SOAPException if there is a SOAP error
217      */
218     public abstract Name createName(
219         String localName,
220         String prefix,
221         String uri)
222         throws SOAPException;
223
224     /**
225      * Creates a new Name object initialized with the
226      * given local name.
227      * <P>
228      * This factory method creates Name objects for use in
229      * situations where it is not practical to use the SOAPEnvelope
230      * abstraction.
231      *
232      * @param localName a String giving the local name
233      * @return a Name object initialized with the given
234      *         local name
235      * @throws SOAPException if there is a SOAP error
236      */
237     public abstract Name createName(String localName) throws SOAPException;
238
239     /**
240      * Creates a new SOAPFactory object that is an instance of
241      * the default implementation (SOAP 1.1),
242      *
243      * This method uses the following ordered lookup procedure to determine the SOAPFactory
244      * implementation class to load:
245      * <UL>
246      * <LI> Use the javax.xml.soap.SOAPFactory system property.
247      * <LI> Use the properties file "lib/jaxm.properties" in the JRE directory. This configuration
248      * file is in standard
249      * java.util.Properties format and contains the fully qualified name of the implementation
250      * class with the key being the
251      * system property defined above.
```

```

249     * <LI> Use the Services API (as detailed in the JAR specification), if available, to
        determine the classname. The Services API
250     * will look for a classname in the file META-INF/services/javax.xml.soap.SOAPFactory in jars
        available to the runtime.
251     * <LI> Use the SAAJMetaFactory instance to locate the SOAPFactory implementation class.
252     * </UL>
253     *
254     * @return a new instance of a <code>SOAPFactory</code>
255     *
256     * @exception SOAPException if there was an error creating the
257     *         default <code>SOAPFactory</code>
258     * @see SAAJMetaFactory
259     */
260     public static SOAPFactory newInstance()
261         throws SOAPException
262     {
263         try {
264             SOAPFactory factory = (SOAPFactory) FactoryFinder.find(
265                 SOAP_FACTORY_PROPERTY, DEFAULT_SOAP_FACTORY, false);
266             if (factory != null)
267                 return factory;
268             return newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);
269         } catch (Exception ex) {
270             throw new SOAPException(
271                 "Unable to create SOAP Factory: " + ex.getMessage());
272         }
273     }
274 }
275
276 /**
277  * Creates a new <code>SOAPFactory</code> object that is an instance of
278  * the specified implementation, this method uses the SAAJMetaFactory to
279  * locate the implementation class and create the SOAPFactory instance.
280  *
281  * @return a new instance of a <code>SOAPFactory</code>
282  *
283  * @param protocol a string constant representing the protocol of the
284  *         specified SOAP factory implementation. May be
285  *         either <code>DYNAMIC_SOAP_PROTOCOL</code>,
286  *         <code>DEFAULT_SOAP_PROTOCOL</code> (which is the same
287  *         as) <code>SOAP_1_1_PROTOCOL</code>, or
288  *         <code>SOAP_1_2_PROTOCOL</code>.
289  *
290  * @exception SOAPException if there was an error creating the
291  *         specified <code>SOAPFactory</code>
292  * @see SAAJMetaFactory
293  * @since SAAJ 1.3
294  */
295     public static SOAPFactory newInstance(String protocol)
296         throws SOAPException {
297         return SAAJMetaFactory.getInstance().newSOAPFactory(protocol);
298     }
299 }

```

**17.22 SOAPFault.java**

Secuencia 183: javax/xml/soap/SoapFault.java

```
1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.soap;
27
28 import java.util.Iterator;
29 import java.util.Locale;
30
31 import javax.xml.namespace.QName;
32
33 /**
34 * An element in the <code>SOAPBody</code> object that contains
35 * error and/or status information. This information may relate to
36 * errors in the <code>SOAPMessage</code> object or to problems
37 * that are not related to the content in the message itself. Problems
38 * not related to the message itself are generally errors in
39 * processing, such as the inability to communicate with an upstream
40 * server.
41 * <P>
42 * Depending on the <code>protocol</code> specified while creating the
43 * <code>MessageFactory</code> instance, a <code>SoapFault</code> has
44 * sub-elements as defined in the SOAP 1.1/SOAP 1.2 specification.
45 */
46 public interface SoapFault extends SoapBodyElement {
47
48     /**
49      * Sets this <code>SoapFault</code> object with the given fault code.
50      *
```

```

51 * <P> Fault codes, which give information about the fault, are defined
52 * in the SOAP 1.1 specification. A fault code is mandatory and must
53 * be of type <code>Name</code>. This method provides a convenient
54 * way to set a fault code. For example,
55 *
56 * <PRE>
57 * SOAPEnvelope se = ...;
58 * // Create a qualified name in the SOAP namespace with a localName
59 * // of "Client". Note that prefix parameter is optional and is null
60 * // here which causes the implementation to use an appropriate prefix.
61 * Name qname = se.createName("Client", null,
62 * SOAPConstants.URI_NS_SOAP_ENVELOPE);
63 * SOAPFault fault = ...;
64 * fault.setFaultCode(qname);
65 * </PRE>
66 * It is preferable to use this method over {@link #setFaultCode(String)}.
67 *
68 * @param faultCodeQName a <code>Name</code> object giving the fault
69 * code to be set. It must be namespace qualified.
70 * @see #getFaultCodeAsName
71 *
72 * @exception SOAPException if there was an error in adding the
73 * <i>faultcode</i> element to the underlying XML tree.
74 *
75 * @since SAAJ 1.2
76 */
77 public void setFaultCode(Name faultCodeQName) throws SOAPException;
78
79 /**
80 * Sets this <code>SOAPFault</code> object with the given fault code.
81 *
82 * It is preferable to use this method over {@link #setFaultCode(Name)}.
83 *
84 * @param faultCodeQName a <code>QName</code> object giving the fault
85 * code to be set. It must be namespace qualified.
86 * @see #getFaultCodeAsQName
87 *
88 * @exception SOAPException if there was an error in adding the
89 * <code>faultcode</code> element to the underlying XML tree.
90 *
91 * @see #setFaultCode(Name)
92 * @see #getFaultCodeAsQName()
93 *
94 * @since SAAJ 1.3
95 */
96 public void setFaultCode(QName faultCodeQName) throws SOAPException;
97
98 /**
99 * Sets this <code>SOAPFault</code> object with the give fault code.
100 * <P>
101 * Fault codes, which given information about the fault, are defined in
102 * the SOAP 1.1 specification. This element is mandatory in SOAP 1.1.
103 * Because the fault code is required to be a QName it is preferable to

```

```

104     * use the {@link #setFaultCode(Name)} form of this method.
105     *
106     * @param faultCode a String giving the fault code to be set.
107     *     It must be of the form "prefix:localName" where the prefix has
108     *     been defined in a namespace declaration.
109     * @see #setFaultCode(Name)
110     * @see #getFaultCode
111     * @see SOAPElement#addNamespaceDeclaration
112     *
113     * @exception SOAPException if there was an error in adding the
114     *     faultCode to the underlying XML tree.
115     */
116     public void setFaultCode(String faultCode) throws SOAPException;
117
118     /**
119     * Gets the mandatory SOAP 1.1 fault code for this
120     * SOAPFault object as a SAAJ Name object.
121     * The SOAP 1.1 specification requires the value of the "faultcode"
122     * element to be of type QName. This method returns the content of the
123     * element as a QName in the form of a SAAJ Name object. This method
124     * should be used instead of the getFaultCode method since
125     * it allows applications to easily access the namespace name without
126     * additional parsing.
127     *
128     * @return a Name representing the faultcode
129     * @see #setFaultCode(Name)
130     *
131     * @since SAAJ 1.2
132     */
133     public Name getFaultCodeAsName();
134
135
136     /**
137     * Gets the fault code for this
138     * SOAPFault object as a QName object.
139     *
140     * @return a QName representing the faultcode
141     *
142     * @see #setFaultCode(QName)
143     *
144     * @since SAAJ 1.3
145     */
146     public QName getFaultCodeAsQName();
147
148     /**
149     * Gets the Subcodes for this SOAPFault as an iterator over
150     * QNames.
151     *
152     * @return an Iterator that accesses a sequence of
153     *     QNames. This Iterator should not support
154     *     the optional remove method. The order in which the
155     *     Subcodes are returned reflects the hierarchy of Subcodes present
156     *     in the fault from top to bottom.

```

```

157      *
158      * @exception UnsupportedOperationException if this message does not
159      *       support the SOAP 1.2 concept of Subcode.
160      *
161      * @since SAAJ 1.3
162      */
163     public Iterator getFaultSubcodes();
164
165     /**
166      * Removes any Subcodes that may be contained by this
167      * <code>SOAPFault</code>. Subsequent calls to
168      * <code>getFaultSubcodes</code> will return an empty iterator until a call
169      * to <code>appendFaultSubcode</code> is made.
170      *
171      * @exception UnsupportedOperationException if this message does not
172      *       support the SOAP 1.2 concept of Subcode.
173      *
174      * @since SAAJ 1.3
175      */
176     public void removeAllFaultSubcodes();
177
178     /**
179      * Adds a Subcode to the end of the sequence of Subcodes contained by this
180      * <code>SOAPFault</code>. Subcodes, which were introduced in SOAP 1.2, are
181      * represented by a recursive sequence of subelements rooted in the
182      * mandatory Code subelement of a SOAP Fault.
183      *
184      * @param subcode a QName containing the Value of the Subcode.
185      *
186      * @exception SOAPException if there was an error in setting the Subcode
187      * @exception UnsupportedOperationException if this message does not
188      *       support the SOAP 1.2 concept of Subcode.
189      *
190      * @since SAAJ 1.3
191      */
192     public void appendFaultSubcode(QName subcode) throws SOAPException;
193
194     /**
195      * Gets the fault code for this <code>SOAPFault</code> object.
196      *
197      * @return a <code>String</code> with the fault code
198      * @see #getFaultCodeAsString
199      * @see #setFaultCode
200      */
201     public String getFaultCode();
202
203     /**
204      * Sets this <code>SOAPFault</code> object with the given fault actor.
205      * <P>
206      * The fault actor is the recipient in the message path who caused the
207      * fault to happen.
208      * <P>
209      * If this <code>SOAPFault</code> supports SOAP 1.2 then this call is

```



```
210     * equivalent to {@link #setFaultRole(String)}
211     *
212     * @param faultActor a <code>String</code> identifying the actor that
213     *     caused this <code>SOAPFault</code> object
214     * @see #getFaultActor
215     *
216     * @exception SOAPException if there was an error in adding the
217     *     <code>faultActor</code> to the underlying XML tree.
218     */
219     public void setFaultActor(String faultActor) throws SOAPException;
220
221     /**
222     * Gets the fault actor for this <code>SOAPFault</code> object.
223     * <P>
224     * If this <code>SOAPFault</code> supports SOAP 1.2 then this call is
225     * equivalent to {@link #getFaultRole()}
226     *
227     * @return a <code>String</code> giving the actor in the message path
228     *     that caused this <code>SOAPFault</code> object
229     * @see #setFaultActor
230     */
231     public String getFaultActor();
232
233     /**
234     * Sets the fault string for this <code>SOAPFault</code> object
235     * to the given string.
236     * <P>
237     * If this
238     * <code>SOAPFault</code> is part of a message that supports SOAP 1.2 then
239     * this call is equivalent to:
240     * <pre>
241     *     addFaultReasonText(faultString, Locale.getDefault());
242     * </pre>
243     *
244     * @param faultString a <code>String</code> giving an explanation of
245     *     the fault
246     * @see #getFaultString
247     *
248     * @exception SOAPException if there was an error in adding the
249     *     <code>faultString</code> to the underlying XML tree.
250     */
251     public void setFaultString(String faultString) throws SOAPException;
252
253     /**
254     * Sets the fault string for this <code>SOAPFault</code> object
255     * to the given string and localized to the given locale.
256     * <P>
257     * If this
258     * <code>SOAPFault</code> is part of a message that supports SOAP 1.2 then
259     * this call is equivalent to:
260     * <pre>
261     *     addFaultReasonText(faultString, locale);
262     * </pre>
```

```

263      *
264      * @param faultString a <code>String</code> giving an explanation of
265      *     the fault
266      * @param locale a {@link java.util.Locale Locale} object indicating
267      *     the native language of the <code>faultString</code>
268      * @see #getFaultString
269      *
270      * @exception SOAPException if there was an error in adding the
271      *     <code>faultString</code> to the underlying XML tree.
272      *
273      * @since SAAJ 1.2
274      */
275      public void setFaultString(String faultString, Locale locale)
276          throws SOAPException;
277
278      /**
279      * Gets the fault string for this <code>SOAPFault</code> object.
280      * <P>
281      * If this
282      * <code>SOAPFault</code> is part of a message that supports SOAP 1.2 then
283      * this call is equivalent to:
284      * <pre>
285      *     String reason = null;
286      *     try {
287      *         reason = (String) getFaultReasonTexts().next();
288      *     } catch (SOAPException e) {}
289      *     return reason;
290      * </pre>
291      *
292      * @return a <code>String</code> giving an explanation of
293      *     the fault
294      * @see #setFaultString(String)
295      * @see #setFaultString(String, Locale)
296      */
297      public String getFaultString();
298
299      /**
300      * Gets the locale of the fault string for this <code>SOAPFault</code>
301      * object.
302      * <P>
303      * If this
304      * <code>SOAPFault</code> is part of a message that supports SOAP 1.2 then
305      * this call is equivalent to:
306      * <pre>
307      *     Locale locale = null;
308      *     try {
309      *         locale = (Locale) getFaultReasonLocales().next();
310      *     } catch (SOAPException e) {}
311      *     return locale;
312      * </pre>
313      *
314      * @return a <code>Locale</code> object indicating the native language of
315      *     the fault string or <code>null</code> if no locale was specified

```

```

316     * @see #setFaultString(String, Locale)
317     *
318     * @since SAAJ 1.2
319     */
320     public Locale getFaultStringLocale();
321
322     /**
323     * Returns true if this <code>SOAPFault</code> has a <code>Detail</code>
324     * subelement and false otherwise. Equivalent to
325     * <code>(getDetail()!=null)</code>.
326     *
327     * @return true if this <code>SOAPFault</code> has a <code>Detail</code>
328     * subelement and false otherwise.
329     *
330     * @since SAAJ 1.3
331     */
332     public boolean hasDetail();
333
334     /**
335     * Returns the optional detail element for this <code>SOAPFault</code>
336     * object.
337     * <P>
338     * A <code>Detail</code> object carries application-specific error
339     * information, the scope of the error information is restricted to
340     * faults in the <code>SOAPBodyElement</code> objects if this is a
341     * SOAP 1.1 Fault.
342     *
343     * @return a <code>Detail</code> object with application-specific
344     *         error information if present, null otherwise
345     */
346     public Detail getDetail();
347
348     /**
349     * Creates an optional <code>Detail</code> object and sets it as the
350     * <code>Detail</code> object for this <code>SOAPFault</code>
351     * object.
352     * <P>
353     * It is illegal to add a detail when the fault already
354     * contains a detail. Therefore, this method should be called
355     * only after the existing detail has been removed.
356     *
357     * @return the new <code>Detail</code> object
358     *
359     * @exception SOAPException if this
360     *         <code>SOAPFault</code> object already contains a
361     *         valid <code>Detail</code> object
362     */
363     public Detail addDetail() throws SOAPException;
364
365     /**
366     * Returns an <code>Iterator</code> over a distinct sequence of
367     * <code>Locale</code>s for which there are associated Reason Text items.
368     * Any of these <code>Locale</code>s can be used in a call to

```

```

369 * <code>getFaultReasonText</code> in order to obtain a localized version
370 * of the Reason Text string.
371 *
372 * @return an <code>Iterator</code> over a sequence of <code>Locale</code>
373 *       objects for which there are associated Reason Text items.
374 *
375 * @exception SOAPException if there was an error in retrieving
376 *       the fault Reason locales.
377 * @exception UnsupportedOperationException if this message does not
378 *       support the SOAP 1.2 concept of Fault Reason.
379 *
380 * @since SAAJ 1.3
381 */
382 public Iterator getFaultReasonLocales() throws SOAPException;
383
384 /**
385 * Returns an <code>Iterator</code> over a sequence of
386 * <code>String</code> objects containing all of the Reason Text items for
387 * this <code>SOAPFault</code>.
388 *
389 * @return an <code>Iterator</code> over env:Fault/env:Reason/env:Text items.
390 *
391 * @exception SOAPException if there was an error in retrieving
392 *       the fault Reason texts.
393 * @exception UnsupportedOperationException if this message does not
394 *       support the SOAP 1.2 concept of Fault Reason.
395 *
396 * @since SAAJ 1.3
397 */
398 public Iterator getFaultReasonTexts() throws SOAPException;
399
400 /**
401 * Returns the Reason Text associated with the given <code>Locale</code>.
402 * If more than one such Reason Text exists the first matching Text is
403 * returned
404 *
405 * @param locale -- the <code>Locale</code> for which a localized
406 *       Reason Text is desired
407 *
408 * @return the Reason Text associated with <code>locale</code>
409 *
410 * @see #getFaultString
411 *
412 * @exception SOAPException if there was an error in retrieving
413 *       the fault Reason text for the specified locale .
414 * @exception UnsupportedOperationException if this message does not
415 *       support the SOAP 1.2 concept of Fault Reason.
416 *
417 * @since SAAJ 1.3
418 */
419 public String getFaultReasonText(Locale locale) throws SOAPException;
420
421 /**

```

```

422     * Appends or replaces a Reason Text item containing the specified
423     * text message and an <i>xml:lang</i> derived from
424     * <code>locale</code>. If a Reason Text item with this
425     * <i>xml:lang</i> already exists its text value will be replaced
426     * with <code>text</code>.
427     * The <code>locale</code> parameter should not be <code>null</code>
428     * <P>
429     * Code sample:
430     *
431     * <PRE>
432     * SOAPFault fault = ...;
433     * fault.addFaultReasonText("Version Mismatch", Locale.ENGLISH);
434     * </PRE>
435     *
436     * @param text -- reason message string
437     * @param locale -- Locale object representing the locale of the message
438     *
439     * @exception SOAPException if there was an error in adding the Reason text
440     * or the <code>locale</code> passed was <code>null</code>.
441     * @exception UnsupportedOperationException if this message does not
442     * support the SOAP 1.2 concept of Fault Reason.
443     *
444     * @since SAAJ 1.3
445     */
446     public void addFaultReasonText(String text, java.util.Locale locale)
447         throws SOAPException;
448
449     /**
450     * Returns the optional Node element value for this
451     * <code>SOAPFault</code> object. The Node element is
452     * optional in SOAP 1.2.
453     *
454     * @return Content of the env:Fault/env:Node element as a String
455     * or <code>null</code> if none
456     *
457     * @exception UnsupportedOperationException if this message does not
458     * support the SOAP 1.2 concept of Fault Node.
459     *
460     * @since SAAJ 1.3
461     */
462     public String getFaultNode();
463
464     /**
465     * Creates or replaces any existing Node element value for
466     * this <code>SOAPFault</code> object. The Node element
467     * is optional in SOAP 1.2.
468     *
469     * @exception SOAPException if there was an error in setting the
470     * Node for this <code>SOAPFault</code> object.
471     * @exception UnsupportedOperationException if this message does not
472     * support the SOAP 1.2 concept of Fault Node.
473     *
474     *

```

```

475     * @since SAAJ 1.3
476     */
477     public void setFaultNode(String uri) throws SOAPException;
478
479     /**
480     * Returns the optional Role element value for this
481     * <code>SOAPFault</code> object. The Role element is
482     * optional in SOAP 1.2.
483     *
484     * @return Content of the env:Fault/env:Role element as a String
485     * or <code>null</code> if none
486     *
487     * @exception UnsupportedOperationException if this message does not
488     * support the SOAP 1.2 concept of Fault Role.
489     *
490     * @since SAAJ 1.3
491     */
492     public String getFaultRole();
493
494     /**
495     * Creates or replaces any existing Role element value for
496     * this <code>SOAPFault</code> object. The Role element
497     * is optional in SOAP 1.2.
498     *
499     * @param uri - the URI of the Role
500     *
501     * @exception SOAPException if there was an error in setting the
502     * Role for this <code>SOAPFault</code> object.
503     *
504     * @exception UnsupportedOperationException if this message does not
505     * support the SOAP 1.2 concept of Fault Role.
506     *
507     * @since SAAJ 1.3
508     */
509     public void setFaultRole(String uri) throws SOAPException;
510
511 }

```

## 17.23 SOAPFaultElement.java

Secuencia 184: javax/xml/soap/SOAPFaultElement.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.soap;
27
28  /**
29   * A representation of the contents in
30   * a <code>SOAPFault</code> object. The <code>Detail</code> interface
31   * is a <code>SOAPFaultElement</code>.
32   * <P>
33   * Content is added to a <code>SOAPFaultElement</code> using the
34   * <code>SOAPElement</code> method <code>addTextNode</code>.
35   */
36  public interface SOAPFaultElement extends SOAPElement {
37  }
```

## 17.24 SOAPHeader.java

Secuencia 185: javax/xml/soap/SOAPHeader.java

```
1  /*
2   * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
```

```

25
26 package javax.xml.soap;
27
28 import java.util.Iterator;
29
30 import javax.xml.namespace.QName;
31
32 /**
33  * A representation of the SOAP header
34  * element. A SOAP header element consists of XML data that affects
35  * the way the application-specific content is processed by the message
36  * provider. For example, transaction semantics, authentication information,
37  * and so on, can be specified as the content of a <code>SOAPHeader</code>
38  * object.
39  * <P>
40  * A <code>SOAPEnvelope</code> object contains an empty
41  * <code>SOAPHeader</code> object by default. If the <code>SOAPHeader</code>
42  * object, which is optional, is not needed, it can be retrieved and deleted
43  * with the following line of code. The variable <i>se</i> is a
44  * <code>SOAPEnvelope</code> object.
45  * <PRE>
46  *     se.getHeader().detachNode();
47  * </PRE>
48  *
49  * A <code>SOAPHeader</code> object is created with the <code>SOAPEnvelope</code>
50  * method <code>addHeader</code>. This method, which creates a new header and adds it
51  * to the envelope, may be called only after the existing header has been removed.
52  *
53  * <PRE>
54  *     se.getHeader().detachNode();
55  *     SOAPHeader sh = se.addHeader();
56  * </PRE>
57  * <P>
58  * A <code>SOAPHeader</code> object can have only <code>SOAPHeaderElement</code>
59  * objects as its immediate children. The method <code>addHeaderElement</code>
60  * creates a new <code>HeaderElement</code> object and adds it to the
61  * <code>SOAPHeader</code> object. In the following line of code, the
62  * argument to the method <code>addHeaderElement</code> is a <code>Name</code>
63  * object that is the name for the new <code>HeaderElement</code> object.
64  * <PRE>
65  *     SOAPHeaderElement shElement = sh.addHeaderElement(name);
66  * </PRE>
67  *
68  * @see SOAPHeaderElement
69  */
70 public interface SOAPHeader extends SOAPElement {
71     /**
72      * Creates a new <code>SOAPHeaderElement</code> object initialized with the
73      * specified name and adds it to this <code>SOAPHeader</code> object.
74      *
75      * @param name a <code>Name</code> object with the name of the new
76      *      <code>SOAPHeaderElement</code> object
77      * @return the new <code>SOAPHeaderElement</code> object that was

```



```

78      *      inserted into this <code>SOAPHeader</code> object
79      * @exception SOAPException if a SOAP error occurs
80      * @see SOAPHeader#addHeaderElement(javax.xml.namespace.QName)
81      */
82      public SOAPHeaderElement addHeaderElement(Name name)
83          throws SOAPException;
84
85      /**
86      * Creates a new <code>SOAPHeaderElement</code> object initialized with the
87      * specified QName and adds it to this <code>SOAPHeader</code> object.
88      *
89      * @param QName a <code>QName</code> object with the QName of the new
90      *      <code>SOAPHeaderElement</code> object
91      * @return the new <code>SOAPHeaderElement</code> object that was
92      *      inserted into this <code>SOAPHeader</code> object
93      * @exception SOAPException if a SOAP error occurs
94      * @see SOAPHeader#addHeaderElement(Name)
95      * @since SAAJ 1.3
96      */
97      public SOAPHeaderElement addHeaderElement(QName qname)
98          throws SOAPException;
99
100     /**
101     * Returns an <code>Iterator</code> over all the <code>SOAPHeaderElement</code> objects
102     * in this <code>SOAPHeader</code> object
103     * that have the specified <i>actor</i> and that have a MustUnderstand attribute
104     * whose value is equivalent to <code>true</code>.
105     * <p>
106     * In SOAP 1.2 the <i>env:actor</i> attribute is replaced by the <i>env:role</i>
107     * attribute, but with essentially the same semantics.
108     *
109     * @param actor a <code>String</code> giving the URI of the <code>actor</code> / <code>role</code>
110     *      code>
111     *      for which to search
112     * @return an <code>Iterator</code> object over all the
113     *      <code>SOAPHeaderElement</code> objects that contain the specified
114     *      <code>actor</code> / <code>role</code> and are marked as MustUnderstand
115     * @see #examineHeaderElements
116     * @see #extractHeaderElements
117     * @see SOAPConstants#URI_SOAP_ACTOR_NEXT
118     *
119     * @since SAAJ 1.2
120     */
121     public Iterator examineMustUnderstandHeaderElements(String actor);
122
123     /**
124     * Returns an <code>Iterator</code> over all the <code>SOAPHeaderElement</code> objects
125     * in this <code>SOAPHeader</code> object
126     * that have the specified <i>actor</i>.
127     *
128     * An <i>actor</i> is a global attribute that indicates the intermediate
129     * parties that should process a message before it reaches its ultimate
130     * receiver. An actor receives the message and processes it before sending

```

```

130 * it on to the next actor. The default actor is the ultimate intended
131 * recipient for the message, so if no actor attribute is included in a
132 * <code>SOAPHeader</code> object, it is sent to the ultimate receiver
133 * along with the message body.
134 * <p>
135 * In SOAP 1.2 the <i>env:actor</i> attribute is replaced by the <i>env:role</i>
136 * attribute, but with essentially the same semantics.
137 *
138 * @param actor a <code>String</code> giving the URI of the <code>actor</code> / <code>role</code>
139 *           for which to search
140 * @return an <code>Iterator</code> object over all the
141 *           <code>SOAPHeaderElement</code> objects that contain the specified
142 *           <code>actor</code> / <code>role</code>
143 * @see #extractHeaderElements
144 * @see SOAPConstants#URI_SOAP_ACTOR_NEXT
145 */
146 public Iterator examineHeaderElements(String actor);
147
148 /**
149 * Returns an <code>Iterator</code> over all the <code>SOAPHeaderElement</code> objects
150 * in this <code>SOAPHeader</code> object
151 * that have the specified <i>actor</i> and detaches them
152 * from this <code>SOAPHeader</code> object.
153 * <p>
154 * This method allows an actor to process the parts of the
155 * <code>SOAPHeader</code> object that apply to it and to remove
156 * them before passing the message on to the next actor.
157 * <p>
158 * In SOAP 1.2 the <i>env:actor</i> attribute is replaced by the <i>env:role</i>
159 * attribute, but with essentially the same semantics.
160 *
161 * @param actor a <code>String</code> giving the URI of the <code>actor</code> / <code>role</code>
162 *           for which to search
163 * @return an <code>Iterator</code> object over all the
164 *           <code>SOAPHeaderElement</code> objects that contain the specified
165 *           <code>actor</code> / <code>role</code>
166 *
167 * @see #examineHeaderElements
168 * @see SOAPConstants#URI_SOAP_ACTOR_NEXT
169 */
170 public Iterator extractHeaderElements(String actor);
171
172 /**
173 * Creates a new NotUnderstood <code>SOAPHeaderElement</code> object initialized
174 * with the specified name and adds it to this <code>SOAPHeader</code> object.
175 * This operation is supported only by SOAP 1.2.
176 *
177 * @param name a <code>QName</code> object with the name of the
178 *           <code>SOAPHeaderElement</code> object that was not understood.
179 * @return the new <code>SOAPHeaderElement</code> object that was
180 *           inserted into this <code>SOAPHeader</code> object

```

```

181     * @exception SOAPException if a SOAP error occurs.
182     * @exception UnsupportedOperationException if this is a SOAP 1.1 Header.
183     * @since SAAJ 1.3
184     */
185     public SOAPHeaderElement addNotUnderstoodHeaderElement(QName name)
186         throws SOAPException;
187
188     /**
189     * Creates a new Upgrade <code>SOAPHeaderElement</code> object initialized
190     * with the specified List of supported SOAP URIs and adds it to this
191     * <code>SOAPHeader</code> object.
192     * This operation is supported on both SOAP 1.1 and SOAP 1.2 header.
193     *
194     * @param supportedSOAPURIs an <code>Iterator</code> object with the URIs of SOAP
195     *     versions supported.
196     * @return the new <code>SOAPHeaderElement</code> object that was
197     *     inserted into this <code>SOAPHeader</code> object
198     * @exception SOAPException if a SOAP error occurs.
199     * @since SAAJ 1.3
200     */
201     public SOAPHeaderElement addUpgradeHeaderElement(Iterator supportedSOAPURIs)
202         throws SOAPException;
203
204     /**
205     * Creates a new Upgrade <code>SOAPHeaderElement</code> object initialized
206     * with the specified array of supported SOAP URIs and adds it to this
207     * <code>SOAPHeader</code> object.
208     * This operation is supported on both SOAP 1.1 and SOAP 1.2 header.
209     *
210     * @param supportedSoapUris an array of the URIs of SOAP versions supported.
211     * @return the new <code>SOAPHeaderElement</code> object that was
212     *     inserted into this <code>SOAPHeader</code> object
213     * @exception SOAPException if a SOAP error occurs.
214     * @since SAAJ 1.3
215     */
216     public SOAPHeaderElement addUpgradeHeaderElement(String[] supportedSoapUris)
217         throws SOAPException;
218
219     /**
220     * Creates a new Upgrade <code>SOAPHeaderElement</code> object initialized
221     * with the specified supported SOAP URI and adds it to this
222     * <code>SOAPHeader</code> object.
223     * This operation is supported on both SOAP 1.1 and SOAP 1.2 header.
224     *
225     * @param supportedSoapUri the URI of SOAP the version that is supported.
226     * @return the new <code>SOAPHeaderElement</code> object that was
227     *     inserted into this <code>SOAPHeader</code> object
228     * @exception SOAPException if a SOAP error occurs.
229     * @since SAAJ 1.3
230     */
231     public SOAPHeaderElement addUpgradeHeaderElement(String supportedSoapUri)
232         throws SOAPException;
233

```

```

234  /**
235   * Returns an Iterator over all the SOAPHeaderElement objects
236   * in this SOAPHeader object.
237   *
238   * @return an Iterator object over all the
239   *         SOAPHeaderElement objects contained by this
240   *         SOAPHeader
241   * @see #extractAllHeaderElements
242   *
243   * @since SAAJ 1.2
244   */
245  public Iterator examineAllHeaderElements();
246
247  /**
248   * Returns an Iterator over all the SOAPHeaderElement objects
249   * in this SOAPHeader object and detaches them
250   * from this SOAPHeader object.
251   *
252   * @return an Iterator object over all the
253   *         SOAPHeaderElement objects contained by this
254   *         SOAPHeader
255   *
256   * @see #extractAllHeaderElements
257   *
258   * @since SAAJ 1.2
259   */
260  public Iterator extractAllHeaderElements();
261
262  }

```

## 17.25 SOAPHeaderElement.java

Secuencia 186: javax/xml/soap/SOAPHeaderElement.java

```

1  /**
2   * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *

```

```

21  *
22  *
23  *
24  */
25
26  package javax.xml.soap;
27
28  /**
29   * An object representing the contents in the SOAP header part of the
30   * SOAP envelope.
31   * The immediate children of a <code>SOAPHeader</code> object can
32   * be represented only as <code>SOAPHeaderElement</code> objects.
33   * <P>
34   * A <code>SOAPHeaderElement</code> object can have other
35   * <code>SOAPElement</code> objects as its children.
36   */
37  public interface SOAPHeaderElement extends SOAPElement {
38
39      /**
40       * Sets the actor associated with this <code>SOAPHeaderElement</code>
41       * object to the specified actor. The default value of an actor is:
42       * <code>SOAPConstants.URI_SOAP_ACTOR_NEXT</code>
43       * <P>
44       * If this <code>SOAPHeaderElement</code> supports SOAP 1.2 then this call is
45       * equivalent to {@link #setRole(String)}
46       *
47       * @param actorURI a <code>String</code> giving the URI of the actor
48       *                to set
49       *
50       * @exception IllegalArgumentException if there is a problem in
51       *                setting the actor.
52       *
53       * @see #getActor
54       */
55      public void setActor(String actorURI);
56
57      /**
58       * Sets the <code>Role</code> associated with this <code>SOAPHeaderElement</code>
59       * object to the specified <code>Role</code>.
60       *
61       * @param uri - the URI of the <code>Role</code>
62       *
63       * @throws SOAPException if there is an error in setting the role
64       *
65       * @exception UnsupportedOperationException if this message does not
66       *                support the SOAP 1.2 concept of Fault Role.
67       *
68       * @since SAAJ 1.3
69       */
70      public void setRole(String uri) throws SOAPException;
71
72      /**
73       * Returns the uri of the <i>actor</i> attribute of this

```

```

74     * <code>SOAPHeaderElement</code>.
75     *<P>
76     * If this <code>SOAPHeaderElement</code> supports SOAP 1.2 then this call is
77     * equivalent to {@link #getRole()}
78     * @return a <code>String</code> giving the URI of the actor
79     * @see #setActor
80     */
81     public String getActor();
82
83     /**
84     * Returns the value of the <i>Role</i> attribute of this
85     * <code>SOAPHeaderElement</code>.
86     *
87     * @return a <code>String</code> giving the URI of the <code>Role</code>
88     *
89     * @exception UnsupportedOperationException if this message does not
90     *         support the SOAP 1.2 concept of Fault Role.
91     *
92     * @since SAAJ 1.3
93     */
94     public String getRole();
95
96     /**
97     * Sets the mustUnderstand attribute for this <code>SOAPHeaderElement</code>
98     * object to be either true or false.
99     * <P>
100    * If the mustUnderstand attribute is on, the actor who receives the
101    * <code>SOAPHeaderElement</code> must process it correctly. This
102    * ensures, for example, that if the <code>SOAPHeaderElement</code>
103    * object modifies the message, that the message is being modified correctly.
104    *
105    * @param mustUnderstand <code>true</code> to set the mustUnderstand
106    *         attribute to true; <code>false</code> to set it to false
107    *
108    * @exception IllegalArgumentException if there is a problem in
109    *         setting the mustUnderstand attribute
110    * @see #getMustUnderstand
111    * @see #setRelay
112    */
113     public void setMustUnderstand(boolean mustUnderstand);
114
115     /**
116     * Returns the boolean value of the mustUnderstand attribute for this
117     * <code>SOAPHeaderElement</code>.
118     *
119     * @return <code>true</code> if the mustUnderstand attribute of this
120     *         <code>SOAPHeaderElement</code> object is turned on; <code>false</code>
121     *         otherwise
122     */
123     public boolean getMustUnderstand();
124
125     /**
126     * Sets the <i>relay</i> attribute for this <code>SOAPHeaderElement</code> to be

```

```

127     * either true or false.
128     * <P>
129     * The SOAP relay attribute is set to true to indicate that the SOAP header
130     * block must be relayed by any node that is targeted by the header block
131     * but not actually process it. This attribute is ignored on header blocks
132     * whose mustUnderstand attribute is set to true or that are targeted at
133     * the ultimate reciever (which is the default). The default value of this
134     * attribute is <code>>false</code>.
135     *
136     * @param relay the new value of the <i>relay</i> attribute
137     *
138     * @exception SOAPException if there is a problem in setting the
139     * relay attribute.
140     * @exception UnsupportedOperationException if this message does not
141     * support the SOAP 1.2 concept of Relay attribute.
142     *
143     * @see #setMustUnderstand
144     * @see #getRelay
145     *
146     * @since SAAJ 1.3
147     */
148     public void setRelay(boolean relay) throws SOAPException;
149
150     /**
151     * Returns the boolean value of the <i>relay</i> attribute for this
152     * <code>SOAPHeaderElement</code>
153     *
154     * @return <code>true</code> if the relay attribute is turned on;
155     * <code>false</code> otherwise
156     *
157     * @exception UnsupportedOperationException if this message does not
158     * support the SOAP 1.2 concept of Relay attribute.
159     *
160     * @see #getMustUnderstand
161     * @see #setRelay
162     *
163     * @since SAAJ 1.3
164     */
165     public boolean getRelay();
166 }

```

## 17.26 SOAPMessage.java

Secuencia 187: javax/xml/soap/SOAPMessage.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```

```
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.soap;
27 import java.io.OutputStream;
28 import java.io.IOException;
29
30 import java.util.Iterator;
31
32 import javax.activation.DataHandler;
33
34 /**
35  * The root class for all SOAP messages. As transmitted on the "wire", a SOAP
36  * message is an XML document or a MIME message whose first body part is an
37  * XML/SOAP document.
38  * <P>
39  * A SOAPMessage object consists of a SOAP part and optionally
40  * one or more attachment parts. The SOAP part for a SOAPMessage
41  * object is a SOAPPart object, which contains information used
42  * for message routing and identification, and which can contain
43  * application-specific content. All data in the SOAP Part of a message must be
44  * in XML format.
45  * <P>
46  * A new SOAPMessage object contains the following by default:
47  * <UL>
48  * <LI>A SOAPPart object
49  * <LI>A SOAPEnvelope object
50  * <LI>A SOAPBody object
51  * <LI>A SOAPHeader object
52  * </UL>
53  * The SOAP part of a message can be retrieved by calling the method SOAPMessage.getSOAPPart
54  * (()).
55  * The SOAPEnvelope object is retrieved from the SOAPPart
56  * object, and the SOAPEnvelope object is used to retrieve the
57  * SOAPBody and SOAPHeader objects.
58  * <PRE>
59  *     SOAPPart sp = message.getSOAPPart();
60  *     SOAPEnvelope se = sp.getEnvelope();
61  *     SOAPBody sb = se.getBody();
```



```

62 *     SOAPHeader sh = se.getHeader();
63 * </PRE>
64 *
65 * <P>
66 * In addition to the mandatory <code>SOAPPart</code> object, a <code>SOAPMessage</code>
67 * object may contain zero or more <code>AttachmentPart</code> objects, each
68 * of which contains application-specific data. The <code>SOAPMessage</code>
69 * interface provides methods for creating <code>AttachmentPart</code>
70 * objects and also for adding them to a <code>SOAPMessage</code> object. A
71 * party that has received a <code>SOAPMessage</code> object can examine its
72 * contents by retrieving individual attachment parts.
73 * <P>
74 * Unlike the rest of a SOAP message, an attachment is not required to be in
75 * XML format and can therefore be anything from simple text to an image file.
76 * Consequently, any message content that is not in XML format must be in an
77 * <code>AttachmentPart</code> object.
78 * <P>
79 * A <code>MessageFactory</code> object may create <code>SOAPMessage</code>
80 * objects with behavior that is specialized to a particular implementation or
81 * application of SAAJ. For instance, a <code>MessageFactory</code> object
82 * may produce <code>SOAPMessage</code> objects that conform to a particular
83 * Profile such as ebXML. In this case a <code>MessageFactory</code> object
84 * might produce <code>SOAPMessage</code> objects that are initialized with
85 * ebXML headers.
86 * <P>
87 * In order to ensure backward source compatibility, methods that are added to
88 * this class after version 1.1 of the SAAJ specification are all concrete
89 * instead of abstract and they all have default implementations. Unless
90 * otherwise noted in the JavaDocs for those methods the default
91 * implementations simply throw an <code>UnsupportedOperationException</code>
92 * and the SAAJ implementation code must override them with methods that
93 * provide the specified behavior. Legacy client code does not have this
94 * restriction, however, so long as there is no claim made that it conforms to
95 * some later version of the specification than it was originally written for.
96 * A legacy class that extends the SOAPMessage class can be compiled and/or run
97 * against succeeding versions of the SAAJ API without modification. If such a
98 * class was correctly implemented then it will continue to behave correctly
99 * relative to the version of the specification against which it was written.
100 *
101 * @see MessageFactory
102 * @see AttachmentPart
103 */
104 public abstract class SOAPMessage {
105     /**
106      * Specifies the character type encoding for the SOAP Message. Valid values
107      * include "utf-8" and "utf-16". See vendor documentation for additional
108      * supported values. The default is "utf-8".
109      *
110      * @see SOAPMessage#setProperty(String, Object) SOAPMessage.setProperty
111      * @since SAAJ 1.2
112      */
113     public static final String CHARACTER_SET_ENCODING =
114         "javax.xml.soap.character-set-encoding";

```

```

115
116 /**
117  * Specifies whether the SOAP Message will contain an XML declaration when
118  * it is sent. The only valid values are "true" and "false". The default is
119  * "false".
120  *
121  * @see SOAPMessage#setProperty(String, Object) SOAPMessage.setProperty
122  * @since SAAJ 1.2
123  */
124 public static final String WRITE_XML_DECLARATION =
125     "javax.xml.soap.write-xml-declaration";
126
127 /**
128  * Sets the description of this <code>SOAPMessage</code> object's
129  * content with the given description.
130  *
131  * @param description a <code>String</code> describing the content of this
132  *     message
133  * @see #getContentDescription
134  */
135 public abstract void setContentDescription(String description);
136
137 /**
138  * Retrieves a description of this <code>SOAPMessage</code> object's
139  * content.
140  *
141  * @return a <code>String</code> describing the content of this
142  *     message or <code>null</code> if no description has been set
143  * @see #setContentDescription
144  */
145 public abstract String getContentDescription();
146
147 /**
148  * Gets the SOAP part of this <code>SOAPMessage</code> object.
149  * <P>
150  * <code>SOAPMessage</code> object contains one or more attachments, the
151  * SOAP Part must be the first MIME body part in the message.
152  *
153  * @return the <code>SOAPPart</code> object for this <code>SOAPMessage</code>
154  *     object
155  */
156 public abstract SOAPPart getSOAPPart();
157
158 /**
159  * Gets the SOAP Body contained in this <code>SOAPMessage</code> object.
160  * <p>
161  *
162  * @return the <code>SOAPBody</code> object contained by this <code>SOAPMessage</code>
163  *     object
164  * @exception SOAPException
165  *     if the SOAP Body does not exist or cannot be retrieved
166  * @since SAAJ 1.2
167  */

```

```
168     public SOAPBody getSOAPBody() throws SOAPException {
169         throw new UnsupportedOperationException("getSOAPBody must be overridden by all subclasses
            of SOAPMessage");
170     }
171
172     /**
173      * Gets the SOAP Header contained in this <code>SOAPMessage</code>
174      * object.
175      * <p>
176      *
177      * @return the <code>SOAPHeader</code> object contained by this <code>SOAPMessage</code>
178      *         object
179      * @exception SOAPException
180      *         if the SOAP Header does not exist or cannot be retrieved
181      * @since SAAJ 1.2
182      */
183     public SOAPHeader getSOAPHeader() throws SOAPException {
184         throw new UnsupportedOperationException("getSOAPHeader must be overridden by all subclasses
            of SOAPMessage");
185     }
186
187     /**
188      * Removes all <code>AttachmentPart</code> objects that have been added
189      * to this <code>SOAPMessage</code> object.
190      * <p>
191      * This method does not touch the SOAP part.
192      */
193     public abstract void removeAllAttachments();
194
195     /**
196      * Gets a count of the number of attachments in this message. This count
197      * does not include the SOAP part.
198      *
199      * @return the number of <code>AttachmentPart</code> objects that are
200      *         part of this <code>SOAPMessage</code> object
201      */
202     public abstract int countAttachments();
203
204     /**
205      * Retrieves all the <code>AttachmentPart</code> objects that are part of
206      * this <code>SOAPMessage</code> object.
207      *
208      * @return an iterator over all the attachments in this message
209      */
210     public abstract Iterator getAttachments();
211
212     /**
213      * Retrieves all the <code>AttachmentPart</code> objects that have header
214      * entries that match the specified headers. Note that a returned
215      * attachment could have headers in addition to those specified.
216      *
217      * @param headers
218      *        a <code>MimeHeaders</code> object containing the MIME
```

```

219         *           headers for which to search
220         * @return an iterator over all attachments that have a header that matches
221         *           one of the given headers
222         */
223     public abstract Iterator getAttachments(MimeHeaders headers);
224
225     /**
226     * Removes all the AttachmentPart objects that have header
227     * entries that match the specified headers. Note that the removed
228     * attachment could have headers in addition to those specified.
229     *
230     * @param headers
231     *         a MimeHeaders object containing the MIME
232     *         headers for which to search
233     * @since SAAJ 1.3
234     */
235     public abstract void removeAttachments(MimeHeaders headers);
236
237
238     /**
239     * Returns an AttachmentPart object that is associated with an
240     * attachment that is referenced by this SOAPElement or
241     * null if no such attachment exists. References can be made
242     * via an href attribute as described in
243     * {@link <a href="http://www.w3.org/TR/SOAP-attachments#SOAPReferenceToAttachments">SOAP
244     *   Messages with Attachments},
245     * or via a single Text child node containing a URI as
246     * described in the WS-I Attachments Profile 1.0 for elements of schema
247     * type {@link <a href="http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html">
248     *   ref:swaRef}. These two mechanisms must be supported.
249     * The support for references via href attribute also implies that
250     * this method should also be supported on an element that is an
251     * <i>xop:Include</i> element (
252     * {@link <a href="http://www.w3.org/2000/xp/Group/3/06/Attachments/XOP.html">XOP</a>}).
253     * other reference mechanisms may be supported by individual
254     * implementations of this standard. Contact your vendor for details.
255     *
256     * @param element The SOAPElement containing the reference to an Attachment
257     * @return the referenced AttachmentPart or null if no such
258     *         AttachmentPart exists or no reference can be
259     *         found in this SOAPElement.
260     * @throws SOAPException if there is an error in the attempt to access the
261     *         attachment
262     *
263     * @since SAAJ 1.3
264     */
265
266     public abstract AttachmentPart getAttachment(SOAPElement element) throws SOAPException;
267
268     /**
269     * Adds the given AttachmentPart object to this SOAPMessage
270     * object. An AttachmentPart object must be created before
271     * it can be added to a message.

```

```

270     *
271     * @param AttachmentPart
272     *         an <code>AttachmentPart</code> object that is to become part
273     *         of this <code>SOAPMessage</code> object
274     * @exception IllegalArgumentException
275     */
276     public abstract void addAttachmentPart(AttachmentPart AttachmentPart);
277
278     /**
279     * Creates a new empty <code>AttachmentPart</code> object. Note that the
280     * method <code>addAttachmentPart</code> must be called with this new
281     * <code>AttachmentPart</code> object as the parameter in order for it to
282     * become an attachment to this <code>SOAPMessage</code> object.
283     *
284     * @return a new <code>AttachmentPart</code> object that can be populated
285     *         and added to this <code>SOAPMessage</code> object
286     */
287     public abstract AttachmentPart createAttachmentPart();
288
289     /**
290     * Creates an <code>AttachmentPart</code> object and populates it using
291     * the given <code>DataHandler</code> object.
292     *
293     * @param dataHandler
294     *         the <code>javax.activation.DataHandler</code> object that
295     *         will generate the content for this <code>SOAPMessage</code>
296     *         object
297     * @return a new <code>AttachmentPart</code> object that contains data
298     *         generated by the given <code>DataHandler</code> object
299     * @exception IllegalArgumentException
300     *         if there was a problem with the specified <code>DataHandler</code>
301     *         object
302     * @see javax.activation.DataHandler
303     * @see javax.activation.DataContentHandler
304     */
305     public AttachmentPart createAttachmentPart(DataHandler dataHandler) {
306         AttachmentPart attachment = createAttachmentPart();
307         attachment.setDataHandler(dataHandler);
308         return attachment;
309     }
310
311     /**
312     * Returns all the transport-specific MIME headers for this <code>SOAPMessage</code>
313     * object in a transport-independent fashion.
314     *
315     * @return a <code>MimeHeaders</code> object containing the <code>MimeHeader</code>
316     *         objects
317     */
318     public abstract MimeHeaders getMimeHeaders();
319
320     /**
321     * Creates an <code>AttachmentPart</code> object and populates it with
322     * the specified data of the specified content type. The type of the

```

```

323 * <code>Object</code> should correspond to the value given for the
324 * <code>Content-Type</code>.
325 *
326 * @param content
327 *     an <code>Object</code> containing the content for the
328 *     <code>AttachmentPart</code> object to be created
329 * @param contentType
330 *     a <code>String</code> object giving the type of content;
331 *     examples are "text/xml", "text/plain", and "image/jpeg"
332 * @return a new <code>AttachmentPart</code> object that contains the
333 *     given data
334 * @exception IllegalArgumentException
335 *     may be thrown if the contentType does not match the type
336 *     of the content object, or if there was no
337 *     <code>DataContentHandler</code> object for the given
338 *     content object
339 * @see javax.activation.DataHandler
340 * @see javax.activation.DataContentHandler
341 */
342 public AttachmentPart createAttachmentPart(
343     Object content,
344     String contentType) {
345     AttachmentPart attachment = createAttachmentPart();
346     attachment.setContent(content, contentType);
347     return attachment;
348 }
349
350 /**
351 * Updates this <code>SOAPMessage</code> object with all the changes that
352 * have been made to it. This method is called automatically when
353 * {@link SOAPMessage#writeTo(OutputStream)} is called. However, if
354 * changes are made to a message that was received or to one that has
355 * already been sent, the method <code>saveChanges</code> needs to be
356 * called explicitly in order to save the changes. The method <code>saveChanges</code>
357 * also generates any changes that can be read back (for example, a
358 * MessageId in profiles that support a message id). All MIME headers in a
359 * message that is created for sending purposes are guaranteed to have
360 * valid values only after <code>saveChanges</code> has been called.
361 * <P>
362 * In addition, this method marks the point at which the data from all
363 * constituent <code>AttachmentPart</code> objects are pulled into the
364 * message.
365 * <P>
366 *
367 * @exception <code>SOAPException</code> if there was a problem saving
368 *     changes to this message.
369 */
370 public abstract void saveChanges() throws SOAPException;
371
372 /**
373 * Indicates whether this <code>SOAPMessage</code> object needs to have
374 * the method <code>saveChanges</code> called on it.
375 *

```

```

376     * @return <code>true</code> if <code>saveChanges</code> needs to be
377     *         called; <code>false</code> otherwise.
378     */
379     public abstract boolean saveRequired();
380
381     /**
382     * Writes this <code>SOAPMessage</code> object to the given output
383     * stream. The externalization format is as defined by the SOAP 1.1 with
384     * Attachments specification.
385     * <P>
386     * If there are no attachments, just an XML stream is written out. For
387     * those messages that have attachments, <code>writeTo</code> writes a
388     * MIME-encoded byte stream.
389     * <P>
390     * Note that this method does not write the transport-specific MIME Headers
391     * of the Message
392     *
393     * @param out
394     *         the <code>OutputStream</code> object to which this <code>SOAPMessage</code>
395     *         object will be written
396     * @exception IOException
397     *         if an I/O error occurs
398     * @exception SOAPException
399     *         if there was a problem in externalizing this SOAP message
400     */
401     public abstract void writeTo(OutputStream out)
402         throws SOAPException, IOException;
403
404     /**
405     * Associates the specified value with the specified property. If there was
406     * already a value associated with this property, the old value is
407     * replaced.
408     * <p>
409     * The valid property names include
410     * {@link SOAPMessage#WRITE\_XML\_DECLARATION} and
411     * {@link SOAPMessage#CHARACTER\_SET\_ENCODING}. All of these standard SAAJ
412     * properties are prefixed by "javax.xml.soap". Vendors may also add
413     * implementation specific properties. These properties must be prefixed
414     * with package names that are unique to the vendor.
415     * <p>
416     * Setting the property <code>WRITE_XML_DECLARATION</code> to <code>"true"</code>
417     * will cause an XML Declaration to be written out at the start of the SOAP
418     * message. The default value of "false" suppresses this declaration.
419     * <p>
420     * The property <code>CHARACTER_SET_ENCODING</code> defaults to the value
421     * <code>"utf-8"</code> which causes the SOAP message to be encoded using
422     * UTF-8. Setting <code>CHARACTER_SET_ENCODING</code> to <code>"utf-16"</code>
423     * causes the SOAP message to be encoded using UTF-16.
424     * <p>
425     * Some implementations may allow encodings in addition to UTF-8 and
426     * UTF-16. Refer to your vendor's documentation for details.
427     *
428     * @param property

```

```

429     *         the property with which the specified value is to be
430     *         associated.
431     * @param value
432     *         the value to be associated with the specified property
433     * @exception SOAPException
434     *         if the property name is not recognized.
435     * @since SAAJ 1.2
436     */
437     public void setProperty(String property, Object value)
438         throws SOAPException {
439         throw new UnsupportedOperationException("setProperty must be overridden by all
440             subclasses of SOAPMessage");
441     }
442     /**
443     * Retrieves value of the specified property.
444     *
445     * @param property
446     *         the name of the property to retrieve
447     * @return the value associated with the named property or <code>null</code>
448     *         if no such property exists.
449     * @exception SOAPException
450     *         if the property name is not recognized.
451     * @since SAAJ 1.2
452     */
453     public Object getProperty(String property) throws SOAPException {
454         throw new UnsupportedOperationException("getProperty must be overridden by all subclasses
455             of SOAPMessage");
456     }

```

## 17.27 SOAPPart.java

Secuencia 188: javax/xml/soap/SOAPPart.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```



```

20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.soap;
27
28  import java.util.Iterator;
29
30  import javax.xml.transform.Source;
31
32  /**
33   * The container for the SOAP-specific portion of a SOAPMessage
34   * object. All messages are required to have a SOAP part, so when a
35   * SOAPMessage object is created, it will automatically
36   * have a SOAPPart object.
37   * <P>
38   * A SOAPPart object is a MIME part and has the MIME headers
39   * Content-Id, Content-Location, and Content-Type. Because the value of
40   * Content-Type must be "text/xml", a SOAPPart object automatically
41   * has a MIME header of Content-Type with its value set to "text/xml".
42   * The value must be "text/xml" because content in the SOAP part of a
43   * message must be in XML format. Content that is not of type "text/xml"
44   * must be in an AttachmentPart object rather than in the
45   * SOAPPart object.
46   * <P>
47   * When a message is sent, its SOAP part must have the MIME header Content-Type
48   * set to "text/xml". Or, from the other perspective, the SOAP part of any
49   * message that is received must have the MIME header Content-Type with a
50   * value of "text/xml".
51   * <P>
52   * A client can access the SOAPPart object of a
53   * SOAPMessage object by
54   * calling the method SOAPMessage.getSOAPPart. The
55   * following line of code, in which message is a
56   * SOAPMessage object, retrieves the SOAP part of a message.
57   * <PRE>
58   *     SOAPPart soapPart = message.getSOAPPart();
59   * </PRE>
60   * <P>
61   * A SOAPPart object contains a SOAPEnvelope object,
62   * which in turn contains a SOAPBody object and a
63   * SOAPHeader object.
64   * The SOAPPart method getEnvelope can be used
65   * to retrieve the SOAPEnvelope object.
66   * <P>
67   */
68  public abstract class SOAPPart implements org.w3c.dom.Document, Node {
69
70      /**
71       * Gets the SOAPEnvelope object associated with this
72       * SOAPPart object. Once the SOAP envelope is obtained, it

```

```

73     * can be used to get its contents.
74     *
75     * @return the <code>SOAPEnvelope</code> object for this
76     *         <code>SOAPPart</code> object
77     * @exception SOAPException if there is a SOAP error
78     */
79     public abstract SOAPEnvelope getEnvelope() throws SOAPException;
80
81     /**
82     * Retrieves the value of the MIME header whose name is "Content-Id".
83     *
84     * @return a <code>String</code> giving the value of the MIME header
85     *         named "Content-Id"
86     * @see #setContentId
87     */
88     public String getContentId() {
89         String[] values = getMimeHeader("Content-Id");
90         if (values != null && values.length > 0)
91             return values[0];
92         return null;
93     }
94
95     /**
96     * Retrieves the value of the MIME header whose name is "Content-Location".
97     *
98     * @return a <code>String</code> giving the value of the MIME header whose
99     *         name is "Content-Location"
100    * @see #setContentLocation
101    */
102    public String getContentLocation() {
103        String[] values = getMimeHeader("Content-Location");
104        if (values != null && values.length > 0)
105            return values[0];
106        return null;
107    }
108
109    /**
110    * Sets the value of the MIME header named "Content-Id"
111    * to the given <code>String</code>.
112    *
113    * @param contentId a <code>String</code> giving the value of the MIME
114    *        header "Content-Id"
115    *
116    * @exception IllegalArgumentException if there is a problem in
117    *        setting the content id
118    * @see #getContentId
119    */
120    public void setContentId(String contentId)
121    {
122        setMimeHeader("Content-Id", contentId);
123    }
124
125    /**
126    * Sets the value of the MIME header "Content-Location"

```

```

126     * to the given <code>String</code>.
127     *
128     * @param contentLocation a <code>String</code> giving the value
129     *       of the MIME
130     *       header "Content-Location"
131     * @exception IllegalArgumentException if there is a problem in
132     *       setting the content location.
133     * @see #getContentLocation
134     */
135     public void setContentLocation(String contentLocation)
136     {
137         setMimeHeader("Content-Location", contentLocation);
138     }
139     /**
140     * Removes all MIME headers that match the given name.
141     *
142     * @param header a <code>String</code> giving the name of the MIME header(s) to
143     *       be removed
144     */
145     public abstract void removeMimeHeader(String header);
146
147     /**
148     * Removes all the <code>MimeHeader</code> objects for this
149     * <code>SOAPEnvelope</code> object.
150     */
151     public abstract void removeAllMimeHeaders();
152
153     /**
154     * Gets all the values of the <code>MimeHeader</code> object
155     * in this <code>SOAPPart</code> object that
156     * is identified by the given <code>String</code>.
157     *
158     * @param name the name of the header; example: "Content-Type"
159     * @return a <code>String</code> array giving all the values for the
160     *       specified header
161     * @see #setMimeHeader
162     */
163     public abstract String[] getMimeHeader(String name);
164
165     /**
166     * Changes the first header entry that matches the given header name
167     * so that its value is the given value, adding a new header with the
168     * given name and value if no
169     * existing header is a match. If there is a match, this method clears
170     * all existing values for the first header that matches and sets the
171     * given value instead. If more than one header has
172     * the given name, this method removes all of the matching headers after
173     * the first one.
174     * <P>
175     * Note that RFC822 headers can contain only US-ASCII characters.
176     *
177     * @param name a <code>String</code> giving the header name
178     *       for which to search

```

```

179     * @param   value   a <code>String</code> giving the value to be set.
180     *               This value will be substituted for the current value(s)
181     *               of the first header that is a match if there is one.
182     *               If there is no match, this value will be the value for
183     *               a new <code>MimeHeader</code> object.
184     *
185     * @exception IllegalArgumentException if there was a problem with
186     *         the specified mime header name or value
187     * @see #getMimeHeader
188     */
189     public abstract void setMimeHeader(String name, String value);
190
191     /**
192     * Creates a <code>MimeHeader</code> object with the specified
193     * name and value and adds it to this <code>SOAPPart</code> object.
194     * If a <code>MimeHeader</code> with the specified name already
195     * exists, this method adds the specified value to the already
196     * existing value(s).
197     * <P>
198     * Note that RFC822 headers can contain only US-ASCII characters.
199     *
200     * @param   name     a <code>String</code> giving the header name
201     * @param   value     a <code>String</code> giving the value to be set
202     *                   or added
203     * @exception IllegalArgumentException if there was a problem with
204     *         the specified mime header name or value
205     */
206     public abstract void addMimeHeader(String name, String value);
207
208     /**
209     * Retrieves all the headers for this <code>SOAPPart</code> object
210     * as an iterator over the <code>MimeHeader</code> objects.
211     *
212     * @return   an <code>Iterator</code> object with all of the Mime
213     *           headers for this <code>SOAPPart</code> object
214     */
215     public abstract Iterator getAllMimeHeaders();
216
217     /**
218     * Retrieves all <code>MimeHeader</code> objects that match a name in
219     * the given array.
220     *
221     * @param   names a <code>String</code> array with the name(s) of the
222     *           MIME headers to be returned
223     * @return   all of the MIME headers that match one of the names in the
224     *           given array, returned as an <code>Iterator</code> object
225     */
226     public abstract Iterator getMatchingMimeHeaders(String[] names);
227
228     /**
229     * Retrieves all <code>MimeHeader</code> objects whose name does
230     * not match a name in the given array.
231     *

```

```

232     * @param names a String array with the name(s) of the
233     *       MIME headers not to be returned
234     * @return all of the MIME headers in this SOAPPart object
235     *       except those that match one of the names in the
236     *       given array. The nonmatching MIME headers are returned as an
237     *       Iterator object.
238     */
239     public abstract Iterator getNonMatchingMimeHeaders(String[] names);
240
241     /**
242     * Sets the content of the SOAPEnvelope object with the data
243     * from the given Source object. This Source
244     * must contain a valid SOAP document.
245     *
246     * @param source the javax.xml.transform.Source object with the
247     *       data to be set
248     *
249     * @exception SOAPException if there is a problem in setting the source
250     * @see #getContent
251     */
252     public abstract void setContent(Source source) throws SOAPException;
253
254     /**
255     * Returns the content of the SOAPEnvelope as a JAXP Source
256     * object.
257     *
258     * @return the content as a javax.xml.transform.Source object
259     *
260     * @exception SOAPException if the implementation cannot convert
261     *       the specified Source object
262     * @see #setContent
263     */
264     public abstract Source getContent() throws SOAPException;
265 }

```

## 17.28 Text.java

Secuencia 189: javax/xml/soap/Text.java

```

1  /*
2  * Copyright (c) 2004, 2012, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.soap;
27
28  /**
29   * A representation of a node whose value is text. A Text object
30   * may represent text that is content or text that is a comment.
31   *
32   */
33  public interface Text extends Node, org.w3c.dom.Text {
34
35      /**
36       * Retrieves whether this Text object represents a comment.
37       *
38       * @return true if this Text object is a
39       *         comment; false otherwise
40       */
41      public boolean isComment();
42  }

```

## 18 Xml Stream (javax.xml.stream package)

### 18.1 EventFilter.java

Secuencia 190: javax/xml/stream/EventFilter.java

```

1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *

```

```

21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  import javax.xml.stream.events.XMLEvent;
32
33  /**
34   * This interface declares a simple filter interface that one can
35   * create to filter XMLEventReaders
36   * @version 1.0
37   * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
38   * @since 1.6
39   */
40  public interface EventFilter {
41      /**
42       * Tests whether this event is part of this stream. This method
43       * will return true if this filter accepts this event and false
44       * otherwise.
45       *
46       * @param event the event to test
47       * @return true if this filter accepts this event, false otherwise
48       */
49      public boolean accept(XMLEvent event);
50  }

```

## 18.2 FactoryConfigurationError.java

Secuencia 191: javax/xml/stream/FactoryConfigurationError.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```

```
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  /**
32  * An error class for reporting factory configuration errors.
33  *
34  * @version 1.0
35  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
36  * @since 1.6
37  */
38  public class FactoryConfigurationError extends Error {
39      private static final long serialVersionUID = -2994412584589975744L;
40
41      Exception nested;
42
43      /**
44       * Default constructor
45       */
46      public FactoryConfigurationError(){}
47
48      /**
49       * Construct an exception with a nested inner exception
50       *
51       * @param e the exception to nest
52       */
53      public FactoryConfigurationError(java.lang.Exception e){
54          nested = e;
55      }
56
57      /**
58       * Construct an exception with a nested inner exception
59       * and a message
60       *
61       * @param e the exception to nest
62       * @param msg the message to report
63       */
64      public FactoryConfigurationError(java.lang.Exception e, java.lang.String msg){
65          super(msg);
66          nested = e;
67      }
68
69      /**
70       * Construct an exception with a nested inner exception
71       * and a message
72       *
```



```
73     * @param msg the message to report
74     * @param e the exception to nest
75     */
76     public FactoryConfigurationError(java.lang.String msg, java.lang.Exception e){
77         super(msg);
78         nested = e;
79     }
80
81     /**
82     * Construct an exception with associated message
83     *
84     * @param msg the message to report
85     */
86     public FactoryConfigurationError(java.lang.String msg) {
87         super(msg);
88     }
89
90     /**
91     * Return the nested exception (if any)
92     *
93     * @return the nested exception or null
94     */
95     public Exception getException() {
96         return nested;
97     }
98
99     /**
100    * use the exception chaining mechanism of JDK1.4
101    */
102    @Override
103    public Throwable getCause() {
104        return nested;
105    }
106
107    /**
108    * Report the message associated with this error
109    *
110    * @return the string value of the message
111    */
112    public String getMessage() {
113        String msg = super.getMessage();
114        if(msg != null)
115            return msg;
116        if(nested != null){
117            msg = nested.getMessage();
118            if(msg == null)
119                msg = nested.getClass().toString();
120        }
121        return msg;
122    }
123
124
125 }
```

## 18.3 FactoryFinder.java

Secuencia 192: javax/xml/stream/FactoryFinder.java

```
1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.stream;
27
28 import java.io.File;
29 import java.security.AccessController;
30 import java.security.PrivilegedAction;
31 import java.util.Iterator;
32 import java.util.Properties;
33 import java.util.ServiceConfigurationError;
34 import java.util.ServiceLoader;
35
36 /**
37  * <p>Implements pluggable streams.</p>
38  *
39  * <p>This class is duplicated for each JAXP subpackage so keep it in
40  * sync. It is package private for secure class loading.</p>
41  *
42  * @author Santiago.PericasGeertsen@sun.com
43  */
44 class FactoryFinder {
45     // Check we have access to package.
46     private static final String DEFAULT_PACKAGE = "com.sun.xml.internal.";
47
48     /**
49      * Internal debug flag.
50      */
51 }
```

```
51     private static boolean debug = false;
52
53     /**
54      * Cache for properties in java.home/lib/jaxp.properties
55      */
56     final private static Properties cacheProps = new Properties();
57
58     /**
59      * Flag indicating if properties from java.home/lib/jaxp.properties
60      * have been cached.
61      */
62     private static volatile boolean firstTime = true;
63
64     /**
65      * Security support class use to check access control before
66      * getting certain system resources.
67      */
68     final private static SecuritySupport ss = new SecuritySupport();
69
70     // Define system property "jaxp.debug" to get output
71     static {
72         // Use try/catch block to support applets, which throws
73         // SecurityException out of this code.
74         try {
75             String val = ss.getSystemProperty("jaxp.debug");
76             // Allow simply setting the prop to turn on debug
77             debug = val != null && !"false".equals(val);
78         }
79         catch (SecurityException se) {
80             debug = false;
81         }
82     }
83
84     private static void dPrint(String msg) {
85         if (debug) {
86             System.err.println("JAXP: " + msg);
87         }
88     }
89
90     /**
91      * Attempt to load a class using the class loader supplied. If that fails
92      * and fall back is enabled, the current (i.e. bootstrap) class loader is
93      * tried.
94      *
95      * If the class loader supplied is <code>null</code>, first try using the
96      * context class loader followed by the current (i.e. bootstrap) class
97      * loader.
98      *
99      * Use bootstrap classLoader if cl = null and useBSClsLoader is true
100     */
101     static private Class getProviderClass(String className, ClassLoader cl,
102         boolean doFallback, boolean useBSClsLoader) throws ClassNotFoundException
103     {
```

```

104     try {
105         if (cl == null) {
106             if (useBSClsLoader) {
107                 return Class.forName(className, false, FactoryFinder.class.getClassLoader());
108             } else {
109                 cl = ss.getContextClassLoader();
110                 if (cl == null) {
111                     throw new ClassNotFoundException();
112                 }
113                 else {
114                     return Class.forName(className, false, cl);
115                 }
116             }
117         }
118         else {
119             return Class.forName(className, false, cl);
120         }
121     }
122     catch (ClassNotFoundException e1) {
123         if (doFallback) {
124             // Use current class loader - should always be bootstrap CL
125             return Class.forName(className, false, FactoryFinder.class.getClassLoader());
126         }
127         else {
128             throw e1;
129         }
130     }
131 }
132
133 /**
134  * Create an instance of a class. Delegates to method
135  * <code>getProviderClass()</code> in order to load the class.
136  *
137  * @param type Base class / Service interface of the factory to
138  *             instantiate.
139  *
140  * @param className Name of the concrete class corresponding to the
141  * service provider
142  *
143  * @param cl <code>ClassLoader</code> used to load the factory class. If <code>null</code>
144  * current <code>Thread</code>'s context classLoader is used to load the factory class.
145  *
146  * @param doFallback True if the current ClassLoader should be tried as
147  * a fallback if the class is not found using cl
148  */
149 static <T> T newInstance(Class<T> type, String className, ClassLoader cl, boolean doFallback)
150     throws FactoryConfigurationError
151 {
152     return newInstance(type, className, cl, doFallback, false);
153 }
154
155 /**
156  * Create an instance of a class. Delegates to method

```

```

157     * <code>getProviderClass()</code> in order to load the class.
158     *
159     * @param type Base class / Service interface of the factory to
160     *           instantiate.
161     *
162     * @param className Name of the concrete class corresponding to the
163     *           service provider
164     *
165     * @param cl <code>ClassLoader</code> used to load the factory class. If <code>null</code>
166     *           current <code>Thread</code>'s context classLoader is used to load the factory class.
167     *
168     * @param doFallback True if the current ClassLoader should be tried as
169     *           a fallback if the class is not found using cl
170     *
171     * @param useBSClsLoader True if cl=null actually meant bootstrap classLoader. This parameter
172     *           is needed since DocumentBuilderFactory/SAXParserFactory defined null as context classLoader.
173     */
174     static <T> T newInstance(Class<T> type, String className, ClassLoader cl,
175                             boolean doFallback, boolean useBSClsLoader)
176         throws FactoryConfigurationError
177     {
178         assert type != null;
179
180         // make sure we have access to restricted packages
181         if (System.getSecurityManager() != null) {
182             if (className != null && className.startsWith(DEFAULT_PACKAGE)) {
183                 cl = null;
184                 useBSClsLoader = true;
185             }
186         }
187
188         try {
189             Class<?> providerClass = getProviderClass(className, cl, doFallback, useBSClsLoader);
190             if (!type.isAssignableFrom(providerClass)) {
191                 throw new ClassCastException(className + " cannot be cast to " + type.getName());
192             }
193             Object instance = providerClass.newInstance();
194             if (debug) { // Extra check to avoid computing cl strings
195                 dPrint("created new instance of " + providerClass +
196                     " using ClassLoader: " + cl);
197             }
198             return type.cast(instance);
199         }
200         catch (ClassNotFoundException x) {
201             throw new FactoryConfigurationError(
202                 "Provider " + className + " not found", x);
203         }
204         catch (Exception x) {
205             throw new FactoryConfigurationError(
206                 "Provider " + className + " could not be instantiated: " + x,
207                 x);
208         }
209     }

```

```

210
211 /**
212  * Finds the implementation Class object in the specified order.
213  *
214  * @return Class object of factory, never null
215  *
216  * @param type          Base class / Service interface  of the
217  *                      factory to find.
218  *
219  * @param fallbackClassName  Implementation class name, if nothing else
220  *                      is found.  Use null to mean no fallback.
221  *
222  * Package private so this code can be shared.
223  */
224 static <T> T find(Class<T> type, String fallbackClassName)
225     throws FactoryConfigurationError
226 {
227     return find(type, type.getName(), null, fallbackClassName);
228 }
229
230 /**
231  * Finds the implementation Class object in the specified order.  Main
232  * entry point.
233  * @return Class object of factory, never null
234  *
235  * @param type          Base class / Service interface  of the
236  *                      factory to find.
237  *
238  * @param factoryId      Name of the factory to find, same as
239  *                      a property name
240  *
241  * @param cl             ClassLoader to be used to load the class, null means to use
242  *                      the bootstrap ClassLoader
243  *
244  * @param fallbackClassName  Implementation class name, if nothing else
245  *                      is found.  Use null to mean no fallback.
246  *
247  * Package private so this code can be shared.
248  */
249 static <T> T find(Class<T> type, String factoryId, ClassLoader cl, String fallbackClassName)
250     throws FactoryConfigurationError
251 {
252     dPrint("find factoryId =" + factoryId);
253
254     // Use the system property first
255     try {
256
257         final String systemProp;
258         if (type.getName().equals(factoryId)) {
259             systemProp = ss.getSystemProperty(factoryId);
260         } else {
261             systemProp = System.getProperty(factoryId);
262         }

```

```

263         if (systemProp != null) {
264             dPrint("found system property, value=" + systemProp);
265             return newInstance(type, systemProp, cl, true);
266         }
267     }
268     catch (SecurityException se) {
269         throw new FactoryConfigurationError(
270             "Failed to read factoryId '" + factoryId + "'", se);
271     }
272
273     // Try read $java.home/lib/stax.properties followed by
274     // $java.home/lib/jaxp.properties if former not present
275     String configFile = null;
276     try {
277         if (firstTime) {
278             synchronized (cacheProps) {
279                 if (firstTime) {
280                     configFile = ss.getProperty("java.home") + File.separator +
281                         "lib" + File.separator + "stax.properties";
282                     File f = new File(configFile);
283                     firstTime = false;
284                     if (ss.isFileExist(f)) {
285                         dPrint("Read properties file "+f);
286                         cacheProps.load(ss.getFileInputStream(f));
287                     }
288                     else {
289                         configFile = ss.getProperty("java.home") + File.separator +
290                             "lib" + File.separator + "jaxp.properties";
291                         f = new File(configFile);
292                         if (ss.isFileExist(f)) {
293                             dPrint("Read properties file "+f);
294                             cacheProps.load(ss.getFileInputStream(f));
295                         }
296                     }
297                 }
298             }
299         }
300         final String factoryClassName = cacheProps.getProperty(factoryId);
301
302         if (factoryClassName != null) {
303             dPrint("found in " + configFile + " value=" + factoryClassName);
304             return newInstance(type, factoryClassName, cl, true);
305         }
306     }
307     catch (Exception ex) {
308         if (debug) ex.printStackTrace();
309     }
310
311     if (type.getName().equals(factoryId)) {
312         // Try Jar Service Provider Mechanism
313         final T provider = findServiceProvider(type, cl);
314         if (provider != null) {
315             return provider;

```

```

316     }
317     } else {
318         // We're in the case where a 'custom' factoryId was provided,
319         // and in every case where that happens, we expect that
320         // fallbackClassName will be null.
321         assert fallbackClassName == null;
322     }
323     if (fallbackClassName == null) {
324         throw new FactoryConfigurationError(
325             "Provider for " + factoryId + " cannot be found", null);
326     }
327
328     dPrint("loaded from fallback value: " + fallbackClassName);
329     return newInstance(type, fallbackClassName, cl, true);
330 }
331
332 /*
333  * Try to find provider using the ServiceLoader API
334  *
335  * @param type Base class / Service interface of the factory to find.
336  *
337  * @return instance of provider class if found or null
338  */
339 private static <T> T findServiceProvider(final Class<T> type, final ClassLoader cl) {
340     try {
341         return AccessController.doPrivileged(new PrivilegedAction<T>() {
342             @Override
343             public T run() {
344                 final ServiceLoader<T> serviceLoader;
345                 if (cl == null) {
346                     //the current thread's context class loader
347                     serviceLoader = ServiceLoader.load(type);
348                 } else {
349                     serviceLoader = ServiceLoader.load(type, cl);
350                 }
351                 final Iterator<T> iterator = serviceLoader.iterator();
352                 if (iterator.hasNext()) {
353                     return iterator.next();
354                 } else {
355                     return null;
356                 }
357             }
358         });
359     } catch (ServiceConfigurationError e) {
360         // It is not possible to wrap an error directly in
361         // FactoryConfigurationError - so we need to wrap the
362         // ServiceConfigurationError in a RuntimeException.
363         // The alternative would be to modify the logic in
364         // FactoryConfigurationError to allow setting a
365         // Throwable as the cause, but that could cause
366         // compatibility issues down the road.
367         final RuntimeException x = new RuntimeException(
368             "Provider for " + type + " cannot be created", e);

```



```

369         final FactoryConfigurationError error =
370             new FactoryConfigurationError(x, x.getMessage());
371         throw error;
372     }
373 }
374
375 }

```

## 18.4 Location.java

Secuencia 193: javax/xml/stream/Location.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream;
30
31 /**
32 * Provides information on the location of an event.
33 *
34 * All the information provided by a Location is optional. For example
35 * an application may only report line numbers.
36 *
37 * @version 1.0
38 * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
39 * @since 1.6
40 */
41 public interface Location {
42     /**

```

```

43  * Return the line number where the current event ends,
44  * returns -1 if none is available.
45  * @return the current line number
46  */
47  int getLineNumber();
48
49  /**
50  * Return the column number where the current event ends,
51  * returns -1 if none is available.
52  * @return the current column number
53  */
54  int getColumnNumber();
55
56  /**
57  * Return the byte or character offset into the input source this location
58  * is pointing to. If the input source is a file or a byte stream then
59  * this is the byte offset into that stream, but if the input source is
60  * a character media then the offset is the character offset.
61  * Returns -1 if there is no offset available.
62  * @return the current offset
63  */
64  int getCharacterOffset();
65
66  /**
67  * Returns the public ID of the XML
68  * @return the public ID, or null if not available
69  */
70  public String getPublicId();
71
72  /**
73  * Returns the system ID of the XML
74  * @return the system ID, or null if not available
75  */
76  public String getSystemId();
77  }

```

## 18.5 SecuritySupport.java

Secuencia 194: javax/xml/stream/SecuritySupport.java

```

1  /*
2  * Copyright (c) 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```

```

15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.stream;
27
28 import java.security.*;
29 import java.net.*;
30 import java.io.*;
31 import java.util.*;
32
33 /**
34  * This class is duplicated for each JAXP subpackage so keep it in sync.
35  * It is package private and therefore is not exposed as part of the JAXP
36  * API.
37  *
38  * Security related methods that only work on J2SE 1.2 and newer.
39  */
40 class SecuritySupport {
41
42     ClassLoader getContextClassLoader() throws SecurityException{
43         return (ClassLoader)
44             AccessController.doPrivileged(new PrivilegedAction() {
45                 public Object run() {
46                     ClassLoader cl = null;
47                     //try {
48                         cl = Thread.currentThread().getContextClassLoader();
49                     //} catch (SecurityException ex) { }
50
51                     if (cl == null)
52                         cl = ClassLoader.getSystemClassLoader();
53
54                     return cl;
55                 }
56             });
57     }
58
59     String getSystemProperty(final String propName) {
60         return (String)
61             AccessController.doPrivileged(new PrivilegedAction() {
62                 public Object run() {
63                     return System.getProperty(propName);
64                 }
65             });
66     }
67 }

```

```

68
69     FileInputStream getFileInputStream(final File file)
70         throws FileNotFoundException
71     {
72         try {
73             return (FileInputStream)
74                 AccessController.doPrivileged(new PrivilegedExceptionAction() {
75                     public Object run() throws FileNotFoundException {
76                         return new FileInputStream(file);
77                     }
78                 });
79         } catch (PrivilegedActionException e) {
80             throw (FileNotFoundException)e.getException();
81         }
82     }
83
84     InputStream getResourceAsStream(final ClassLoader cl,
85                                     final String name)
86     {
87         return (InputStream)
88             AccessController.doPrivileged(new PrivilegedAction() {
89                 public Object run() {
90                     InputStream ris;
91                     if (cl == null) {
92                         ris = Object.class.getResourceAsStream(name);
93                     } else {
94                         ris = cl.getResourceAsStream(name);
95                     }
96                     return ris;
97                 }
98             });
99     }
100
101     boolean doesFileExist(final File f) {
102         return ((Boolean)
103             AccessController.doPrivileged(new PrivilegedAction() {
104                 public Object run() {
105                     return new Boolean(f.exists());
106                 }
107             })).booleanValue();
108     }
109
110 }

```

## 18.6 StreamFilter.java

Secuencia 195: javax/xml/stream/StreamFilter.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *

```

```

7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29 package javax.xml.stream;
30
31 /**
32  * This interface declares a simple filter interface that one can
33  * create to filter XMLStreamReaders
34  * @version 1.0
35  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
36  * @since 1.6
37  */
38 public interface StreamFilter {
39
40     /**
41      * Tests whether the current state is part of this stream. This method
42      * will return true if this filter accepts this event and false otherwise.
43      *
44      * The method should not change the state of the reader when accepting
45      * a state.
46      *
47      * @param reader the event to test
48      * @return true if this filter accepts this event, false otherwise
49      */
50     public boolean accept(XMLStreamReader reader);
51 }

```

## 18.7 XMLEventFactory.java

Secuencia 196: javax/xml/stream/XMLEventFactory.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *

```

```
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26  * Copyright (c) 2009, 2013, by Oracle Corporation. All Rights Reserved.
27  */
28
29 package javax.xml.stream;
30 import java.util.Iterator;
31 import javax.xml.namespace.NamespaceContext;
32 import javax.xml.namespace.QName;
33 import javax.xml.stream.events.*;
34 /**
35  * This interface defines a utility class for creating instances of
36  * XMLEvents
37  * @version 1.2
38  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
39  * @see javax.xml.stream.events.StartElement
40  * @see javax.xml.stream.events.EndElement
41  * @see javax.xml.stream.events.ProcessingInstruction
42  * @see javax.xml.stream.events.Comment
43  * @see javax.xml.stream.events.Characters
44  * @see javax.xml.stream.events.StartDocument
45  * @see javax.xml.stream.events.EndDocument
46  * @see javax.xml.stream.events.DTD
47  * @since 1.6
48  */
49 public abstract class XMLEventFactory {
50     protected XMLEventFactory(){}
51
52     static final String JAXPFACORYID = "javax.xml.stream.XMLEventFactory";
53     static final String DEFAULTIMPL = "com.sun.xml.internal.stream.events.XMLEventFactoryImpl";
54
55
56     /**
57      * Creates a new instance of the factory in exactly the same manner as the
```

```

58  * {@link #newFactory()} method.
59  * @throws FactoryConfigurationError if an instance of this factory cannot be loaded
60  */
61  public static XMLEventFactory newInstance()
62      throws FactoryConfigurationError
63  {
64      return FactoryFinder.find(XMLEventFactory.class, DEFAULTIMPL);
65  }
66
67  /**
68   * Create a new instance of the factory.
69   * <p>
70   * This static method creates a new factory instance.
71   * This method uses the following ordered lookup procedure to determine
72   * the XMLEventFactory implementation class to load:
73   * </p>
74   * <ul>
75   * <li>
76   * Use the javax.xml.stream.XMLEventFactory system property.
77   * </li>
78   * <li>
79   * Use the properties file "lib/stax.properties" in the JRE directory.
80   * This configuration file is in standard java.util.Properties format
81   * and contains the fully qualified name of the implementation class
82   * with the key being the system property defined above.
83   * </li>
84   * <li>
85   * Use the service-provider loading facilities, defined by the
86   * {@link java.util.ServiceLoader} class, to attempt to locate and load an
87   * implementation of the service using the {@linkplain
88   * java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
89   * the service-provider loading facility will use the {@linkplain
90   * java.lang.Thread#getContextClassLoader() current thread's context class loader}
91   * to attempt to load the service. If the context class
92   * loader is null, the {@linkplain
93   * ClassLoader#getSystemClassLoader() system class loader} will be used.
94   * </li>
95   * <li>
96   * Otherwise, the system-default implementation is returned.
97   * </li>
98   * </ul>
99   * <p>
100  * Once an application has obtained a reference to a XMLEventFactory it
101  * can use the factory to configure and obtain stream instances.
102  * </p>
103  * <p>
104  * Note that this is a new method that replaces the deprecated newInstance() method.
105  * No changes in behavior are defined by this replacement method relative to
106  * the deprecated method.
107  * </p>
108  * @throws FactoryConfigurationError in case of {@linkplain
109  * java.util.ServiceConfigurationError service configuration error} or if
110  * the implementation is not available or cannot be instantiated.

```

```

111  */
112  public static XMLEventFactory newFactory()
113      throws FactoryConfigurationError
114  {
115      return FactoryFinder.find(XMLEventFactory.class, DEFAULTIMPL);
116  }
117
118  /**
119   * Create a new instance of the factory
120   *
121   * @param factoryId      Name of the factory to find, same as
122   *                       a property name
123   * @param classLoader    classLoader to use
124   * @return the factory implementation
125   * @throws FactoryConfigurationError if an instance of this factory cannot be loaded
126   *
127   * @deprecated This method has been deprecated to maintain API consistency.
128   *             All newInstance methods have been replaced with corresponding
129   *             newFactory methods. The replacement {@link
130   *             #newFactory(java.lang.String, java.lang.ClassLoader)}
131   *             method defines no changes in behavior.
132   */
133  public static XMLEventFactory newInstance(String factoryId,
134      ClassLoader classLoader)
135      throws FactoryConfigurationError {
136      //do not fallback if given classloader can't find the class, throw exception
137      return FactoryFinder.find(XMLEventFactory.class, factoryId, classLoader, null);
138  }
139
140  /**
141   * Create a new instance of the factory.
142   * If the classLoader argument is null, then the ContextClassLoader is used.
143   * <p>
144   * This method uses the following ordered lookup procedure to determine
145   * the XMLEventFactory implementation class to load:
146   * </p>
147   * <ul>
148   * <li>
149   * Use the value of the system property identified by {@code factoryId}.
150   * </li>
151   * <li>
152   * Use the properties file "lib/stax.properties" in the JRE directory.
153   * This configuration file is in standard java.util.Properties format
154   * and contains the fully qualified name of the implementation class
155   * with the key being the given {@code factoryId}.
156   * </li>
157   * <li>
158   * If {@code factoryId} is "javax.xml.stream.XMLEventFactory",
159   * use the service-provider loading facilities, defined by the
160   * {@link java.util.ServiceLoader} class, to attempt to {@linkplain
161   * java.util.ServiceLoader#load(java.lang.Class, java.lang.ClassLoader) locate and load}
162   * an implementation of the service using the specified {@code ClassLoader}.
163   * If {@code classLoader} is null, the {@linkplain

```



```

164 * java.util.ServiceLoader#load(java.lang.Class) default loading mechanism} will apply:
165 * That is, the service-provider loading facility will use the {@linkplain
166 * java.lang.Thread#getContextClassLoader() current thread's context class loader}
167 * to attempt to load the service. If the context class
168 * loader is null, the {@linkplain
169 * ClassLoader#getSystemClassLoader() system class loader} will be used.
170 * </li>
171 * <li>
172 * Otherwise, throws a {@link FactoryConfigurationError}.
173 * </li>
174 * </ul>
175 *
176 * <p>
177 * Note that this is a new method that replaces the deprecated
178 * {@link #newInstance(java.lang.String, java.lang.ClassLoader)
179 * newInstance(String factoryId, ClassLoader classLoader)} method.
180 * No changes in behavior are defined by this replacement method relative
181 * to the deprecated method.
182 * </p>
183 *
184 * @apiNote The parameter factoryId defined here is inconsistent with that
185 * of other JAXP factories where the first parameter is fully qualified
186 * factory class name that provides implementation of the factory.
187 *
188 * @param factoryId      Name of the factory to find, same as
189 *                       a property name
190 * @param classLoader    classLoader to use
191 * @return the factory implementation
192 * @throws FactoryConfigurationError in case of {@linkplain
193 * java.util.ServiceConfigurationError service configuration error} or if
194 * the implementation is not available or cannot be instantiated.
195 */
196 public static XMLEventFactory newFactory(String factoryId,
197                                         ClassLoader classLoader)
198     throws FactoryConfigurationError {
199     //do not fallback if given classloader can't find the class, throw exception
200     return FactoryFinder.find(XMLEventFactory.class, factoryId, classLoader, null);
201 }
202
203 /**
204 * This method allows setting of the Location on each event that
205 * is created by this factory. The values are copied by value into
206 * the events created by this factory. To reset the location
207 * information set the location to null.
208 * @param location the location to set on each event created
209 */
210 public abstract void setLocation(Location location);
211
212 /**
213 * Create a new Attribute
214 * @param prefix the prefix of this attribute, may not be null
215 * @param namespaceURI the attribute value is set to this value, may not be null
216 * @param localName the local name of the XML name of the attribute, localName cannot be null

```

```
217     * @param value the attribute value to set, may not be null
218     * @return the Attribute with specified values
219     */
220     public abstract Attribute createAttribute(String prefix, String namespaceURI, String localName,
221         String value);
222
223     /**
224     * Create a new Attribute
225     * @param localName the local name of the XML name of the attribute, localName cannot be null
226     * @param value the attribute value to set, may not be null
227     * @return the Attribute with specified values
228     */
229     public abstract Attribute createAttribute(String localName, String value);
230
231     /**
232     * Create a new Attribute
233     * @param name the qualified name of the attribute, may not be null
234     * @param value the attribute value to set, may not be null
235     * @return the Attribute with specified values
236     */
237     public abstract Attribute createAttribute(QName name, String value);
238
239     /**
240     * Create a new default Namespace
241     * @param namespaceURI the default namespace uri
242     * @return the Namespace with the specified value
243     */
244     public abstract Namespace createNamespace(String namespaceURI);
245
246     /**
247     * Create a new Namespace
248     * @param prefix the prefix of this namespace, may not be null
249     * @param namespaceUri the attribute value is set to this value, may not be null
250     * @return the Namespace with the specified values
251     */
252     public abstract Namespace createNamespace(String prefix, String namespaceUri);
253
254     /**
255     * Create a new StartElement. Namespaces can be added to this StartElement
256     * by passing in an Iterator that walks over a set of Namespace interfaces.
257     * Attributes can be added to this StartElement by passing an iterator
258     * that walks over a set of Attribute interfaces.
259     *
260     * @param name the qualified name of the attribute, may not be null
261     * @param attributes an optional unordered set of objects that
262     * implement Attribute to add to the new StartElement, may be null
263     * @param namespaces an optional unordered set of objects that
264     * implement Namespace to add to the new StartElement, may be null
265     * @return an instance of the requested StartElement
266     */
267     public abstract StartElement createStartElement(QName name,
268         Iterator attributes,
269         Iterator namespaces);
```

[illegible]

```
322         String localName,
323         Iterator attributes,
324         Iterator namespaces,
325         NamespaceContext context
326     );
327
328 /**
329  * Create a new EndElement
330  * @param name the qualified name of the EndElement
331  * @param namespaces an optional unordered set of objects that
332  * implement Namespace that have gone out of scope, may be null
333  * @return an instance of the requested EndElement
334  */
335 public abstract EndElement createEndElement(QName name,
336         Iterator namespaces);
337
338 /**
339  * Create a new EndElement
340  * @param namespaceUri the uri of the QName of the new StartElement
341  * @param localName the local name of the QName of the new StartElement
342  * @param prefix the prefix of the QName of the new StartElement
343  * @return an instance of the requested EndElement
344  */
345 public abstract EndElement createEndElement(String prefix,
346         String namespaceUri,
347         String localName);
348
349 /**
350  * Create a new EndElement
351  * @param namespaceUri the uri of the QName of the new StartElement
352  * @param localName the local name of the QName of the new StartElement
353  * @param prefix the prefix of the QName of the new StartElement
354  * @param namespaces an unordered set of objects that implement
355  * Namespace that have gone out of scope, may be null
356  * @return an instance of the requested EndElement
357  */
358 public abstract EndElement createEndElement(String prefix,
359         String namespaceUri,
360         String localName,
361         Iterator namespaces);
362
363 /**
364  * Create a Characters event, this method does not check if the content
365  * is all whitespace. To create a space event use #createSpace(String)
366  * @param content the string to create
367  * @return a Characters event
368  */
369 public abstract Characters createCharacters(String content);
370
371 /**
372  * Create a Characters event with the CData flag set to true
373  * @param content the string to create
374  * @return a Characters event
375  */
```

```
375 public abstract Characters createCDATA(String content);
376
377 /**
378  * Create a Characters event with the isSpace flag set to true
379  * @param content the content of the space to create
380  * @return a Characters event
381  */
382 public abstract Characters createSpace(String content);
383 /**
384  * Create an ignorable space
385  * @param content the space to create
386  * @return a Characters event
387  */
388 public abstract Characters createIgnorableSpace(String content);
389
390 /**
391  * Creates a new instance of a StartDocument event
392  * @return a StartDocument event
393  */
394 public abstract StartDocument createStartDocument();
395
396 /**
397  * Creates a new instance of a StartDocument event
398  *
399  * @param encoding the encoding style
400  * @param version the XML version
401  * @param standalone the status of standalone may be set to "true" or "false"
402  * @return a StartDocument event
403  */
404 public abstract StartDocument createStartDocument(String encoding,
405                                                    String version,
406                                                    boolean standalone);
407
408 /**
409  * Creates a new instance of a StartDocument event
410  *
411  * @param encoding the encoding style
412  * @param version the XML version
413  * @return a StartDocument event
414  */
415 public abstract StartDocument createStartDocument(String encoding,
416                                                    String version);
417
418 /**
419  * Creates a new instance of a StartDocument event
420  *
421  * @param encoding the encoding style
422  * @return a StartDocument event
423  */
424 public abstract StartDocument createStartDocument(String encoding);
425
426 /**
427  * Creates a new instance of an EndDocument event
```

```

428     * @return an EndDocument event
429     */
430     public abstract EndDocument createEndDocument();
431
432     /** Creates a new instance of a EntityReference event
433     *
434     * @param name The name of the reference
435     * @param declaration the declaration for the event
436     * @return an EntityReference event
437     */
438     public abstract EntityReference createEntityReference(String name,
439                                                         EntityDeclaration declaration);
440
441     /**
442     * Create a comment
443     * @param text The text of the comment
444     * a Comment event
445     */
446     public abstract Comment createComment(String text);
447
448     /**
449     * Create a processing instruction
450     * @param target The target of the processing instruction
451     * @param data The text of the processing instruction
452     * @return a ProcessingInstruction event
453     */
454     public abstract ProcessingInstruction createProcessingInstruction(String target,
455                                                                     String data);
456
457     /**
458     * Create a document type definition event
459     * This string contains the entire document type declaration that matches
460     * the doctype decl in the XML 1.0 specification
461     * @param dtd the text of the document type definition
462     * @return a DTD event
463     */
464     public abstract DTD createDTD(String dtd);
465 }

```

## 18.8 XMLEventReader.java

Secuencia 197: javax/xml/stream/XMLEventReader.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  import javax.xml.stream.events.XMLEvent;
32
33  import java.util.Iterator;
34
35  /**
36  *
37  * This is the top level interface for parsing XML Events. It provides
38  * the ability to peek at the next event and returns configuration
39  * information through the property interface.
40  *
41  * @version 1.0
42  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
43  * @see XMLInputFactory
44  * @see XMLEventWriter
45  * @since 1.6
46  */
47  public interface XMLEventReader extends Iterator {
48      /**
49       * Get the next XMLEvent
50       * @see XMLEvent
51       * @throws XMLStreamException if there is an error with the underlying XML.
52       * @throws NoSuchElementException iteration has no more elements.
53       */
54      public XMLEvent nextEvent() throws XMLStreamException;
55
56      /**
57       * Check if there are more events.
58       * Returns true if there are more events and false otherwise.
59       * @return true if the event reader has more events, false otherwise
60       */
61      public boolean hasNext();
62
63      /**
64       * Check the next XMLEvent without reading it from the stream.
65       * Returns null if the stream is at EOF or has no more XMLEvents.
```

```

66     * A call to peek() will be equal to the next return of next().
67     * @see XMLEvent
68     * @throws XMLStreamException
69     */
70     public XMLEvent peek() throws XMLStreamException;
71
72     /**
73     * Reads the content of a text-only element. Precondition:
74     * the current event is START_ELEMENT. Postcondition:
75     * The current event is the corresponding END_ELEMENT.
76     * @throws XMLStreamException if the current event is not a START_ELEMENT
77     * or if a non text element is encountered
78     */
79     public String getElementText() throws XMLStreamException;
80
81     /**
82     * Skips any insignificant space events until a START_ELEMENT or
83     * END_ELEMENT is reached. If anything other than space characters are
84     * encountered, an exception is thrown. This method should
85     * be used when processing element-only content because
86     * the parser is not able to recognize ignorable whitespace if
87     * the DTD is missing or not interpreted.
88     * @throws XMLStreamException if anything other than space characters are encountered
89     */
90     public XMLEvent nextTag() throws XMLStreamException;
91
92     /**
93     * Get the value of a feature/property from the underlying implementation
94     * @param name The name of the property
95     * @return The value of the property
96     * @throws IllegalArgumentException if the property is not supported
97     */
98     public Object getProperty(java.lang.String name) throws java.lang.IllegalArgumentException;
99
100    /**
101    * Frees any resources associated with this Reader. This method does not close the
102    * underlying input source.
103    * @throws XMLStreamException if there are errors freeing associated resources
104    */
105    public void close() throws XMLStreamException;
106 }

```

## 18.9 XMLEventWriter.java

Secuencia 198: javax/xml/stream/XMLEventWriter.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *

```



```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29 package javax.xml.stream;
30
31 import javax.xml.stream.events.*;
32 import javax.xml.stream.util.XMLEventConsumer;
33 import javax.xml.namespace.NamespaceContext;
34
35 /**
36  *
37  * This is the top level interface for writing XML documents.
38  *
39  * Instances of this interface are not required to validate the
40  * form of the XML.
41  *
42  * @version 1.0
43  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
44  * @see XMLEventReader
45  * @see javax.xml.stream.events.XMLEvent
46  * @see javax.xml.stream.events.Characters
47  * @see javax.xml.stream.events.ProcessingInstruction
48  * @see javax.xml.stream.events.StartElement
49  * @see javax.xml.stream.events.EndElement
50  * @since 1.6
51  */
52 public interface XMLEventWriter extends XMLEventConsumer {
53
54     /**
55      * Writes any cached events to the underlying output mechanism
56      * @throws XMLStreamException
57      */
58     public void flush() throws XMLStreamException;
59
60     /**
61      * Frees any resources associated with this stream
```

```

62     * @throws XMLStreamException
63     */
64     public void close() throws XMLStreamException;
65
66     /**
67     * Add an event to the output stream
68     * Adding a START_ELEMENT will open a new namespace scope that
69     * will be closed when the corresponding END_ELEMENT is written.
70     * <table border="2" rules="all" cellpadding="4">
71     *     <thead>
72     *         <tr>
73     *             <th align="center" colspan="2">
74     *                 Required and optional fields for events added to the writer
75     *             </th>
76     *         </tr>
77     *     </thead>
78     *     <tbody>
79     *         <tr>
80     *             <th>Event Type</th>
81     *             <th>Required Fields</th>
82     *             <th>Optional Fields</th>
83     *             <th>Required Behavior</th>
84     *         </tr>
85     *         <tr>
86     *             <td> START_ELEMENT </td>
87     *             <td> QName name </td>
88     *             <td> namespaces , attributes </td>
89     *             <td> A START_ELEMENT will be written by writing the name,
90     *             namespaces, and attributes of the event in XML 1.0 valid
91     *             syntax for START_ELEMENTS.
92     *             The name is written by looking up the prefix for
93     *             the namespace uri. The writer can be configured to
94     *             respect prefixes of QNames. If the writer is respecting
95     *             prefixes it must use the prefix set on the QName. The
96     *             default behavior is to lookup the value for the prefix
97     *             on the EventWriter's internal namespace context.
98     *             Each attribute (if any)
99     *             is written using the behavior specified in the attribute
100     *             section of this table. Each namespace (if any) is written
101     *             using the behavior specified in the namespace section of this
102     *             table.
103     *         </td>
104     *     </tr>
105     *     <tr>
106     *         <td> END_ELEMENT </td>
107     *         <td> QName name </td>
108     *         <td> None </td>
109     *         <td> A well formed END_ELEMENT tag is written.
110     *         The name is written by looking up the prefix for
111     *         the namespace uri. The writer can be configured to
112     *         respect prefixes of QNames. If the writer is respecting
113     *         prefixes it must use the prefix set on the QName. The
114     *         default behavior is to lookup the value for the prefix

```

```

115 *      on the EventWriter's internal namespace context.
116 *      If the END_ELEMENT name does not match the START_ELEMENT
117 *      name an XMLStreamException is thrown.
118 *      </td>
119 *    </tr>
120 *    <tr>
121 *      <td> ATTRIBUTE </td>
122 *      <td> QName name , String value </td>
123 *      <td> QName type </td>
124 *      <td> An attribute is written using the same algorithm
125 *      to find the lexical form as used in START_ELEMENT.
126 *      The default is to use double quotes to wrap attribute
127 *      values and to escape any double quotes found in the
128 *      value. The type value is ignored.
129 *    </td>
130 *  </tr>
131 *  <tr>
132 *    <td> NAMESPACE </td>
133 *    <td> String prefix, String namespaceURI,
134 *    boolean isDefaultNamespaceDeclaration
135 *  </td>
136 *    <td> None </td>
137 *    <td> A namespace declaration is written. If the
138 *    namespace is a default namespace declaration
139 *    (isDefault is true) then xmlns="$namespaceURI"
140 *    is written and the prefix is optional. If
141 *    isDefault is false, the prefix must be declared
142 *    and the writer must prepend xmlns to the prefix
143 *    and write out a standard prefix declaration.
144 *  </td>
145 *  </tr>
146 *  <tr>
147 *    <td> PROCESSING_INSTRUCTION </td>
148 *    <td> None</td>
149 *    <td> String target, String data</td>
150 *    <td> The data does not need to be present and may be
151 *    null. Target is required and many not be null.
152 *    The writer
153 *    will write data section
154 *    directly after the target,
155 *    enclosed in appropriate XML 1.0 syntax
156 *  </td>
157 *  </tr>
158 *  <tr>
159 *    <td> COMMENT </td>
160 *    <td> None </td>
161 *    <td> String comment </td>
162 *    <td> If the comment is present (not null) it is written, otherwise an
163 *    an empty comment is written
164 *  </td>
165 *  </tr>
166 *  <tr>
167 *    <td> START_DOCUMENT </td>

```

```

168 *      <td> None </td>
169 *      <td> String encoding , boolean standalone, String version </td>
170 *      <td> A START_DOCUMENT event is not required to be written to the
171 *          stream. If present the attributes are written inside
172 *          the appropriate XML declaration syntax
173 *      </td>
174 *  </tr>
175 *  <tr>
176 *      <td> END_DOCUMENT </td>
177 *      <td> None </td>
178 *      <td> None </td>
179 *      <td> Nothing is written to the output </td>
180 *  </tr>
181 *  <tr>
182 *      <td> DTD </td>
183 *      <td> String DocumentTypeDefinition </td>
184 *      <td> None </td>
185 *      <td> The DocumentTypeDefinition is written to the output </td>
186 *  </tr>
187 * </tbody>
188 * </table>
189 * @param event the event to be added
190 * @throws XMLStreamException
191 */
192 public void add(XMLEvent event) throws XMLStreamException;
193
194 /**
195 * Adds an entire stream to an output stream,
196 * calls next() on the inputStream argument until hasNext() returns false
197 * This should be treated as a convenience method that will
198 * perform the following loop over all the events in an
199 * event reader and call add on each event.
200 *
201 * @param reader the event stream to add to the output
202 * @throws XMLStreamException
203 */
204
205 public void add(XMLEventReader reader) throws XMLStreamException;
206
207 /**
208 * Gets the prefix the uri is bound to
209 * @param uri the uri to look up
210 * @throws XMLStreamException
211 */
212 public String getPrefix(String uri) throws XMLStreamException;
213
214 /**
215 * Sets the prefix the uri is bound to. This prefix is bound
216 * in the scope of the current START_ELEMENT / END_ELEMENT pair.
217 * If this method is called before a START_ELEMENT has been written
218 * the prefix is bound in the root scope.
219 * @param prefix the prefix to bind to the uri
220 * @param uri the uri to bind to the prefix

```

```

221     * @throws XMLStreamException
222     */
223     public void setPrefix(String prefix, String uri) throws XMLStreamException;
224
225     /**
226     * Binds a URI to the default namespace
227     * This URI is bound
228     * in the scope of the current START_ELEMENT / END_ELEMENT pair.
229     * If this method is called before a START_ELEMENT has been written
230     * the uri is bound in the root scope.
231     * @param uri the uri to bind to the default namespace
232     * @throws XMLStreamException
233     */
234     public void setDefaultNamespace(String uri) throws XMLStreamException;
235
236     /**
237     * Sets the current namespace context for prefix and uri bindings.
238     * This context becomes the root namespace context for writing and
239     * will replace the current root namespace context. Subsequent calls
240     * to setPrefix and setDefaultNamespace will bind namespaces using
241     * the context passed to the method as the root context for resolving
242     * namespaces.
243     * @param context the namespace context to use for this writer
244     * @throws XMLStreamException
245     */
246     public void setNamespaceContext(NamespaceContext context)
247         throws XMLStreamException;
248
249     /**
250     * Returns the current namespace context.
251     * @return the current namespace context
252     */
253     public NamespaceContext getNamespaceContext();
254
255
256 }

```

## 18.10 XMLInputFactory.java

Secuencia 199: javax/xml/stream/XMLInputFactory.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```

14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009, 2013, by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  import javax.xml.stream.util.XMLEventAllocator;
32  import javax.xml.transform.Source;
33
34  /**
35   * Defines an abstract implementation of a factory for getting streams.
36   *
37   * The following table defines the standard properties of this specification.
38   * Each property varies in the level of support required by each implementation.
39   * The level of support required is described in the 'Required' column.
40   *
41   * <table border="2" rules="all" cellpadding="4">
42   *   <thead>
43   *     <tr>
44   *       <th align="center" colspan="5">
45   *         Configuration parameters
46   *       </th>
47   *     </tr>
48   *   </thead>
49   *   <tbody>
50   *     <tr>
51   *       <th>Property Name</th>
52   *       <th>Behavior</th>
53   *       <th>Return type</th>
54   *       <th>Default Value</th>
55   *       <th>Required</th>
56   *     </tr>
57   *     <tr><td>javax.xml.stream.isValidating</td><td>Turns on/off implementation specific DTD
58   *       validation</td><td>Boolean</td><td>False</td><td>No</td></tr>
59   *     <tr><td>javax.xml.stream.isNamespaceAware</td><td>Turns on/off namespace processing for XML 1.0
60   *       support</td><td>Boolean</td><td>True</td><td>True (required) / False (optional)</td></tr>
61   *     <tr><td>javax.xml.stream.isCoalescing</td><td>Requires the processor to coalesce adjacent
62   *       character data</td><td>Boolean</td><td>False</td><td>Yes</td></tr>
63   *     <tr><td>javax.xml.stream.isReplacingEntityReferences</td><td>replace internal entity references
64   *       with their replacement text and report them as characters</td><td>Boolean</td><td>True</td><td>
65   *       Yes</td></tr>
66   *     <tr><td>javax.xml.stream.isSupportingExternalEntities</td><td>Resolve external parsed entities</

```

```

        td><td>Boolean</td><td>Unspecified</td><td>Yes</td></tr>
62 *<tr><td>javax.xml.stream.supportDTD</td><td>Use this property to request processors that do not
    support DTDs</td><td>Boolean</td><td>True</td><td>Yes</td></tr>
63 *<tr><td>javax.xml.stream.reporter</td><td>sets/gets the impl of the XMLReporter </td><td>javax.
    xml.stream.XMLReporter</td><td>Null</td><td>Yes</td></tr>
64 *<tr><td>javax.xml.stream.resolver</td><td>sets/gets the impl of the XMLResolver interface</td><td>
    >javax.xml.stream.XMLResolver</td><td>Null</td><td>Yes</td></tr>
65 *<tr><td>javax.xml.stream allocator</td><td>sets/gets the impl of the XMLEventAllocator interface
    </td><td>javax.xml.stream.util.XMLEventAllocator</td><td>Null</td><td>Yes</td></tr>
66 * </tbody>
67 * </table>
68 *
69 *
70 * @version 1.2
71 * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
72 * @see XMLOutputFactory
73 * @see XMLEventReader
74 * @see XMLStreamReader
75 * @see EventFilter
76 * @see XMLReporter
77 * @see XMLResolver
78 * @see javax.xml.stream.util.XMLEventAllocator
79 * @since 1.6
80 */
81
82 public abstract class XMLInputFactory {
83     /**
84      * The property used to turn on/off namespace support,
85      * this is to support XML 1.0 documents,
86      * only the true setting must be supported
87      */
88     public static final String IS_NAMESPACE_AWARE=
89         "javax.xml.stream.isNamespaceAware";
90
91     /**
92      * The property used to turn on/off implementation specific validation
93      */
94     public static final String IS_VALIDATING=
95         "javax.xml.stream.isValidating";
96
97     /**
98      * The property that requires the parser to coalesce adjacent character data sections
99      */
100     public static final String IS_COALESCING=
101         "javax.xml.stream.isCoalescing";
102
103     /**
104      * Requires the parser to replace internal
105      * entity references with their replacement
106      * text and report them as characters
107      */
108     public static final String IS_REPLACING_ENTITY_REFERENCES=
109         "javax.xml.stream.isReplacingEntityReferences";

```

```
110
111 /**
112  * The property that requires the parser to resolve external parsed entities
113  */
114 public static final String IS_SUPPORTING_EXTERNAL_ENTITIES=
115     "javax.xml.stream.isSupportingExternalEntities";
116
117 /**
118  * The property that requires the parser to support DTDs
119  */
120 public static final String SUPPORT_DTD=
121     "javax.xml.stream.supportDTD";
122
123 /**
124  * The property used to
125  * set/get the implementation of the XMLReporter interface
126  */
127 public static final String REPORTER=
128     "javax.xml.stream.reporter";
129
130 /**
131  * The property used to set/get the implementation of the XMLResolver
132  */
133 public static final String RESOLVER=
134     "javax.xml.stream.resolver";
135
136 /**
137  * The property used to set/get the implementation of the allocator
138  */
139 public static final String ALLOCATOR=
140     "javax.xml.stream.allocator";
141
142 static final String DEFAULTIMPL = "com.sun.xml.internal.stream.XMLInputFactoryImpl";
143
144 protected XMLInputFactory(){}
145
146 /**
147  * Creates a new instance of the factory in exactly the same manner as the
148  * {@link #newFactory()} method.
149  * @throws FactoryConfigurationError if an instance of this factory cannot be loaded
150  */
151 public static XMLInputFactory newInstance()
152     throws FactoryConfigurationError
153 {
154     return FactoryFinder.find(XMLInputFactory.class, DEFAULTIMPL);
155 }
156
157 /**
158  * Create a new instance of the factory.
159  * <p>
160  * This static method creates a new factory instance.
161  * This method uses the following ordered lookup procedure to determine
162  * the XMLInputFactory implementation class to load:
```



```

163 * </p>
164 * <ul>
165 * <li>
166 *   Use the javax.xml.stream.XMLInputFactory system property.
167 * </li>
168 * <li>
169 *   Use the properties file "lib/stax.properties" in the JRE directory.
170 *     This configuration file is in standard java.util.Properties format
171 *     and contains the fully qualified name of the implementation class
172 *     with the key being the system property defined above.
173 * </li>
174 * <li>
175 *   Use the service-provider loading facilities, defined by the
176 *   {@link java.util.ServiceLoader} class, to attempt to locate and load an
177 *   implementation of the service using the {@linkplain
178 *   java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
179 *   the service-provider loading facility will use the {@linkplain
180 *   java.lang.Thread#getContextClassLoader() current thread's context class loader}
181 *   to attempt to load the service. If the context class
182 *   loader is null, the {@linkplain
183 *   ClassLoader#getClassLoader() system class loader} will be used.
184 * </li>
185 * <li>
186 *   Otherwise, the system-default implementation is returned.
187 * </li>
188 * </ul>
189 * <p>
190 *   Once an application has obtained a reference to a XMLInputFactory it
191 *   can use the factory to configure and obtain stream instances.
192 * </p>
193 * <p>
194 *   Note that this is a new method that replaces the deprecated newInstance() method.
195 *   No changes in behavior are defined by this replacement method relative to
196 *   the deprecated method.
197 * </p>
198 * @throws FactoryConfigurationError in case of {@linkplain
199 *   java.util.ServiceConfigurationError service configuration error} or if
200 *   the implementation is not available or cannot be instantiated.
201 */
202 public static XMLInputFactory newFactory()
203     throws FactoryConfigurationError
204 {
205     return FactoryFinder.find(XMLInputFactory.class, DEFAULTIMPL);
206 }
207
208 /**
209 * Create a new instance of the factory
210 *
211 * @param factoryId      Name of the factory to find, same as
212 *                       a property name
213 * @param classLoader    classLoader to use
214 * @return the factory implementation
215 * @throws FactoryConfigurationError if an instance of this factory cannot be loaded

```

```

216 *
217 * @deprecated This method has been deprecated to maintain API consistency.
218 * All newInstance methods have been replaced with corresponding
219 * newFactory methods. The replacement {@link
220 * #newFactory(java.lang.String, java.lang.ClassLoader)} method
221 * defines no changes in behavior.
222 */
223 public static XMLInputFactory newInstance(String factoryId,
224     ClassLoader classLoader)
225     throws FactoryConfigurationError {
226     //do not fallback if given classloader can't find the class, throw exception
227     return FactoryFinder.find(XMLInputFactory.class, factoryId, classLoader, null);
228 }
229
230 /**
231 * Create a new instance of the factory.
232 * If the classLoader argument is null, then the ContextClassLoader is used.
233 * <p>
234 * This method uses the following ordered lookup procedure to determine
235 * the XMLInputFactory implementation class to load:
236 * </p>
237 * <ul>
238 * <li>
239 * Use the value of the system property identified by {@code factoryId}.
240 * </li>
241 * <li>
242 * Use the properties file "lib/stax.properties" in the JRE directory.
243 * This configuration file is in standard java.util.Properties format
244 * and contains the fully qualified name of the implementation class
245 * with the key being the given {@code factoryId}.
246 * </li>
247 * <li>
248 * If {@code factoryId} is "javax.xml.stream.XMLInputFactory",
249 * use the service-provider loading facilities, defined by the
250 * {@link java.util.ServiceLoader} class, to attempt to {@linkplain
251 * java.util.ServiceLoader#load(java.lang.Class, java.lang.ClassLoader) locate and load}
252 * an implementation of the service using the specified {@code ClassLoader}.
253 * If {@code classLoader} is null, the {@linkplain
254 * java.util.ServiceLoader#load(java.lang.Class) default loading mechanism} will apply:
255 * That is, the service-provider loading facility will use the {@linkplain
256 * java.lang.Thread#getContextClassLoader() current thread's context class loader}
257 * to attempt to load the service. If the context class
258 * loader is null, the {@linkplain
259 * ClassLoader#getSystemClassLoader() system class loader} will be used.
260 * </li>
261 * <li>
262 * Otherwise, throws a {@link FactoryConfigurationError}.
263 * </li>
264 * </ul>
265 *
266 * <p>
267 * Note that this is a new method that replaces the deprecated
268 * {@link #newInstance(java.lang.String, java.lang.ClassLoader)}

```

```
269 * newInstance(String factoryId, ClassLoader classLoader)} method.
270 * No changes in behavior are defined by this replacement method relative
271 * to the deprecated method.
272 * </p>
273 *
274 * @apiNote The parameter factoryId defined here is inconsistent with that
275 * of other JAXP factories where the first parameter is fully qualified
276 * factory class name that provides implementation of the factory.
277 *
278 * @param factoryId      Name of the factory to find, same as
279 *                        a property name
280 * @param classLoader    classLoader to use
281 * @return the factory implementation
282 * @throws FactoryConfigurationError in case of {@linkplain
283 *      java.util.ServiceConfigurationError service configuration error} or if
284 *      the implementation is not available or cannot be instantiated.
285 * @throws FactoryConfigurationError if an instance of this factory cannot be loaded
286 */
287 public static XMLInputFactory newFactory(String factoryId,
288     ClassLoader classLoader)
289     throws FactoryConfigurationError {
290     //do not fallback if given classloader can't find the class, throw exception
291     return FactoryFinder.find(XMLInputFactory.class, factoryId, classLoader, null);
292 }
293
294 /**
295 * Create a new XMLStreamReader from a reader
296 * @param reader the XML data to read from
297 * @throws XMLStreamException
298 */
299 public abstract XMLStreamReader createXMLStreamReader(java.io.Reader reader)
300     throws XMLStreamException;
301
302 /**
303 * Create a new XMLStreamReader from a JAXP source. This method is optional.
304 * @param source the source to read from
305 * @throws UnsupportedOperationException if this method is not
306 * supported by this XMLInputFactory
307 * @throws XMLStreamException
308 */
309 public abstract XMLStreamReader createXMLStreamReader(Source source)
310     throws XMLStreamException;
311
312 /**
313 * Create a new XMLStreamReader from a java.io.InputStream
314 * @param stream the InputStream to read from
315 * @throws XMLStreamException
316 */
317 public abstract XMLStreamReader createXMLStreamReader(java.io.InputStream stream)
318     throws XMLStreamException;
319
320 /**
321 * Create a new XMLStreamReader from a java.io.InputStream
```

```
322     * @param stream the InputStream to read from
323     * @param encoding the character encoding of the stream
324     * @throws XMLStreamException
325     */
326     public abstract XMLStreamReader createXMLStreamReader(java.io.InputStream stream, String encoding
327         )
328         throws XMLStreamException;
329
330     /**
331     * Create a new XMLStreamReader from a java.io.InputStream
332     * @param systemId the system ID of the stream
333     * @param stream the InputStream to read from
334     */
335     public abstract XMLStreamReader createXMLStreamReader(String systemId, java.io.InputStream stream
336         )
337         throws XMLStreamException;
338
339     /**
340     * Create a new XMLStreamReader from a java.io.InputStream
341     * @param systemId the system ID of the stream
342     * @param reader the InputStream to read from
343     */
344     public abstract XMLStreamReader createXMLStreamReader(String systemId, java.io.Reader reader)
345         throws XMLStreamException;
346
347     /**
348     * Create a new XMLEventReader from a reader
349     * @param reader the XML data to read from
350     * @throws XMLStreamException
351     */
352     public abstract XMLEventReader createXMLEventReader(java.io.Reader reader)
353         throws XMLStreamException;
354
355     /**
356     * Create a new XMLEventReader from a reader
357     * @param systemId the system ID of the input
358     * @param reader the XML data to read from
359     * @throws XMLStreamException
360     */
361     public abstract XMLEventReader createXMLEventReader(String systemId, java.io.Reader reader)
362         throws XMLStreamException;
363
364     /**
365     * Create a new XMLEventReader from an XMLStreamReader. After being used
366     * to construct the XMLEventReader instance returned from this method
367     * the XMLStreamReader must not be used.
368     * @param reader the XMLStreamReader to read from (may not be modified)
369     * @return a new XMLEventReader
370     * @throws XMLStreamException
371     */
372     public abstract XMLEventReader createXMLEventReader(XMLStreamReader reader)
373         throws XMLStreamException;
```

```
373 /**
374  * Create a new XMLEventReader from a JAXP source.
375  * Support of this method is optional.
376  * @param source the source to read from
377  * @throws UnsupportedOperationException if this method is not
378  * supported by this XMLInputFactory
379  */
380 public abstract XMLEventReader createXMLEventReader(Source source)
381     throws XMLStreamException;
382
383 /**
384  * Create a new XMLEventReader from a java.io.InputStream
385  * @param stream the InputStream to read from
386  * @throws XMLStreamException
387  */
388 public abstract XMLEventReader createXMLEventReader(java.io.InputStream stream)
389     throws XMLStreamException;
390
391 /**
392  * Create a new XMLEventReader from a java.io.InputStream
393  * @param stream the InputStream to read from
394  * @param encoding the character encoding of the stream
395  * @throws XMLStreamException
396  */
397 public abstract XMLEventReader createXMLEventReader(java.io.InputStream stream, String encoding)
398     throws XMLStreamException;
399
400 /**
401  * Create a new XMLEventReader from a java.io.InputStream
402  * @param systemId the system ID of the stream
403  * @param stream the InputStream to read from
404  * @throws XMLStreamException
405  */
406 public abstract XMLEventReader createXMLEventReader(String systemId, java.io.InputStream stream)
407     throws XMLStreamException;
408
409 /**
410  * Create a filtered reader that wraps the filter around the reader
411  * @param reader the reader to filter
412  * @param filter the filter to apply to the reader
413  * @throws XMLStreamException
414  */
415 public abstract XMLStreamReader createFilteredReader(XMLStreamReader reader, StreamFilter filter)
416     throws XMLStreamException;
417
418 /**
419  * Create a filtered event reader that wraps the filter around the event reader
420  * @param reader the event reader to wrap
421  * @param filter the filter to apply to the event reader
422  * @throws XMLStreamException
423  */
424 public abstract XMLEventReader createFilteredReader(XMLEventReader reader, EventFilter filter)
425     throws XMLStreamException;
```

```

426
427 /**
428  * The resolver that will be set on any XMLStreamReader or XMLEventReader created
429  * by this factory instance.
430  */
431 public abstract XMLResolver getXMLResolver();
432
433 /**
434  * The resolver that will be set on any XMLStreamReader or XMLEventReader created
435  * by this factory instance.
436  * @param resolver the resolver to use to resolve references
437  */
438 public abstract void setXMLResolver(XMLResolver resolver);
439
440 /**
441  * The reporter that will be set on any XMLStreamReader or XMLEventReader created
442  * by this factory instance.
443  */
444 public abstract XMLReporter getXMLReporter();
445
446 /**
447  * The reporter that will be set on any XMLStreamReader or XMLEventReader created
448  * by this factory instance.
449  * @param reporter the resolver to use to report non fatal errors
450  */
451 public abstract void setXMLReporter(XMLReporter reporter);
452
453 /**
454  * Allows the user to set specific feature/property on the underlying
455  * implementation. The underlying implementation is not required to support
456  * every setting of every property in the specification and may use
457  * IllegalArgumentException to signal that an unsupported property may not be
458  * set with the specified value.
459  * <p>
460  * All implementations that implement JAXP 1.5 or newer are required to
461  * support the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} property.
462  * </p>
463  * <ul>
464  *   <li>
465  *     <p>
466  *       Access to external DTDs, external Entity References is restricted to the
467  *       protocols specified by the property. If access is denied during parsing
468  *       due to the restriction of this property, {@link javax.xml.stream.XMLStreamException}
469  *       will be thrown by the {@link javax.xml.stream.XMLStreamReader#next()} or
470  *       {@link javax.xml.stream.XMLEventReader#nextEvent()} method.
471  *     </p>
472  *   </li>
473  * </ul>
474  * @param name The name of the property (may not be null)
475  * @param value The value of the property
476  * @throws java.lang.IllegalArgumentException if the property is not supported
477  */
478 public abstract void setProperty(java.lang.String name, Object value)

```

```

479     throws java.lang.IllegalArgumentException;
480
481     /**
482     * Get the value of a feature/property from the underlying implementation
483     * @param name The name of the property (may not be null)
484     * @return The value of the property
485     * @throws IllegalArgumentException if the property is not supported
486     */
487     public abstract Object getProperty(java.lang.String name)
488         throws java.lang.IllegalArgumentException;
489
490
491     /**
492     * Query the set of properties that this factory supports.
493     *
494     * @param name The name of the property (may not be null)
495     * @return true if the property is supported and false otherwise
496     */
497     public abstract boolean isPropertySupported(String name);
498
499     /**
500     * Set a user defined event allocator for events
501     * @param allocator the user defined allocator
502     */
503     public abstract void setEventAllocator(XMLEventAllocator allocator);
504
505     /**
506     * Gets the allocator used by streams created with this factory
507     */
508     public abstract XMLEventAllocator getEventAllocator();
509
510 }

```

## 18.11 XMLOutputFactory.java

Secuencia 200: javax/xml/stream/XMLOutputFactory.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```

18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009, 2013, by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  import javax.xml.transform.Result;
32
33  /**
34   * Defines an abstract implementation of a factory for
35   * getting XMLEventWriters and XMLStreamWriters.
36   *
37   * The following table defines the standard properties of this specification.
38   * Each property varies in the level of support required by each implementation.
39   * The level of support required is described in the 'Required' column.
40   *
41   * 




```



```

70  * on the current StartElement for
71  * any attribute that does not
72  * currently have a namespace declaration in scope. If the StartElement
73  * has a uri but no prefix specified a prefix will be assigned, if the prefix
74  * has not been declared in a parent of the current StartElement it will be declared
75  * on the current StartElement. If the defaultNamespace is bound and in scope
76  * and the default namespace matches the URI of the attribute or StartElement
77  * QName no prefix will be assigned.</p>
78  *
79  * <p>If an element or attribute name has a prefix, but is not
80  * bound to any namespace URI, then the prefix will be removed
81  * during serialization.</p>
82  *
83  * <p>If element and/or attribute names in the same start or
84  * empty-element tag are bound to different namespace URIs and
85  * are using the same prefix then the element or the first
86  * occurring attribute retains the original prefix and the
87  * following attributes have their prefixes replaced with a
88  * new prefix that is bound to the namespace URIs of those
89  * attributes. </p>
90  *
91  * <p>If an element or attribute name uses a prefix that is
92  * bound to a different URI than that inherited from the
93  * namespace context of the parent of that element and there
94  * is no namespace declaration in the context of the current
95  * element then such a namespace declaration is added. </p>
96  *
97  * <p>If an element or attribute name is bound to a prefix and
98  * there is a namespace declaration that binds that prefix
99  * to a different URI then that namespace declaration is
100  * either removed if the correct mapping is inherited from
101  * the parent context of that element, or changed to the
102  * namespace URI of the element or attribute using that prefix.</p>
103  *
104  * @version 1.2
105  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
106  * @see XMLInputFactory
107  * @see XMLEventWriter
108  * @see XMLStreamWriter
109  * @since 1.6
110  */
111 public abstract class XMLOutputFactory {
112     /**
113      * Property used to set prefix defaulting on the output side
114      */
115     public static final String IS_REPAIRING_NAMESPACES=
116         "javax.xml.stream.isRepairingNamespaces";
117
118     static final String DEFAULT_IMPL = "com.sun.xml.internal.stream.XMLOutputFactoryImpl";
119
120     protected XMLOutputFactory(){}
121
122     /**

```

```
123 * Creates a new instance of the factory in exactly the same manner as the
124 * {@link #newInstance()} method.
125 * @throws FactoryConfigurationError if an instance of this factory cannot be loaded
126 */
127 public static XMLOutputFactory newInstance()
128     throws FactoryConfigurationError
129 {
130     return FactoryFinder.find(XMLOutputFactory.class, DEFAULTIMPL);
131 }
132
133 /**
134  * Create a new instance of the factory.
135  * <p>
136  * This static method creates a new factory instance. This method uses the
137  * following ordered lookup procedure to determine the XMLOutputFactory
138  * implementation class to load:
139  * </p>
140  * <ul>
141  * <li>
142  *     Use the javax.xml.stream.XMLOutputFactory system property.
143  * </li>
144  * <li>
145  *     Use the properties file "lib/stax.properties" in the JRE directory.
146  *     This configuration file is in standard java.util.Properties format
147  *     and contains the fully qualified name of the implementation class
148  *     with the key being the system property defined above.
149  * </li>
150  * <li>
151  *     Use the service-provider loading facilities, defined by the
152  *     {@link java.util.ServiceLoader} class, to attempt to locate and load an
153  *     implementation of the service using the {@linkplain
154  *     java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
155  *     the service-provider loading facility will use the {@linkplain
156  *     java.lang.Thread#getContextClassLoader() current thread's context class loader}
157  *     to attempt to load the service. If the context class
158  *     loader is null, the {@linkplain
159  *     ClassLoader#getSystemClassLoader() system class loader} will be used.
160  * </li>
161  * <li>
162  *     Otherwise, the system-default implementation is returned.
163  * </li>
164  * <p>
165  * Once an application has obtained a reference to a XMLOutputFactory it
166  * can use the factory to configure and obtain stream instances.
167  * </p>
168  * <p>
169  * Note that this is a new method that replaces the deprecated newInstance() method.
170  * No changes in behavior are defined by this replacement method relative to the
171  * deprecated method.
172  * </p>
173  * @throws FactoryConfigurationError in case of {@linkplain
174  *     java.util.ServiceConfigurationError service configuration error} or if
175  *     the implementation is not available or cannot be instantiated.
```

```

176  */
177  public static XMLOutputFactory newFactory()
178      throws FactoryConfigurationError
179  {
180      return FactoryFinder.find(XMLOutputFactory.class, DEFAULTIMPL);
181  }
182
183  /**
184   * Create a new instance of the factory.
185   *
186   * @param factoryId      Name of the factory to find, same as
187   *                       a property name
188   * @param classLoader    classLoader to use
189   * @return the factory implementation
190   * @throws FactoryConfigurationError if an instance of this factory cannot be loaded
191   *
192   * @deprecated This method has been deprecated because it returns an
193   *             instance of XMLInputFactory, which is of the wrong class.
194   *             Use the new method {@link #newFactory(java.lang.String,
195   *             java.lang.ClassLoader)} instead.
196   */
197  public static XMLInputFactory newInstance(String factoryId,
198      ClassLoader classLoader)
199      throws FactoryConfigurationError {
200      //do not fallback if given classloader can't find the class, throw exception
201      return FactoryFinder.find(XMLInputFactory.class, factoryId, classLoader, null);
202  }
203
204  /**
205   * Create a new instance of the factory.
206   * If the classLoader argument is null, then the ContextClassLoader is used.
207   * <p>
208   * This method uses the following ordered lookup procedure to determine
209   * the XMLOutputFactory implementation class to load:
210   * </p>
211   * <ul>
212   * <li>
213   * Use the value of the system property identified by {@code factoryId}.
214   * </li>
215   * <li>
216   * Use the properties file "lib/stax.properties" in the JRE directory.
217   * This configuration file is in standard java.util.Properties format
218   * and contains the fully qualified name of the implementation class
219   * with the key being the given {@code factoryId}.
220   * </li>
221   * <li>
222   * If {@code factoryId} is "javax.xml.stream.XMLOutputFactory",
223   * use the service-provider loading facilities, defined by the
224   * {@link java.util.ServiceLoader} class, to attempt to {@linkplain
225   * java.util.ServiceLoader#load(java.lang.Class, java.lang.ClassLoader) locate and load}
226   * an implementation of the service using the specified {@code ClassLoader}.
227   * If {@code classLoader} is null, the {@linkplain
228   * java.util.ServiceLoader#load(java.lang.Class) default loading mechanism} will apply:

```

```

229 * That is, the service-provider loading facility will use the {@linkplain
230 * java.lang.Thread#getContextClassLoader() current thread's context class loader}
231 * to attempt to load the service. If the context class
232 * loader is null, the {@linkplain
233 * ClassLoader#getSystemClassLoader() system class loader} will be used.
234 * </li>
235 * <li>
236 * Otherwise, throws a {@link FactoryConfigurationError}.
237 * </li>
238 * </ul>
239 *
240 * @apiNote The parameter factoryId defined here is inconsistent with that
241 * of other JAXP factories where the first parameter is fully qualified
242 * factory class name that provides implementation of the factory.
243 *
244 * <p>
245 * Note that this is a new method that replaces the deprecated
246 * {@link #newInstance(java.lang.String, java.lang.ClassLoader)
247 * newInstance(String factoryId, ClassLoader classLoader)} method.
248 * The original method was incorrectly defined to return XMLInputFactory.
249 * </p>
250 *
251 * @param factoryId      Name of the factory to find, same as
252 *                       a property name
253 * @param classLoader    classLoader to use
254 * @return the factory implementation
255 * @throws FactoryConfigurationError in case of {@linkplain
256 * java.util.ServiceConfigurationError service configuration error} or if
257 * the implementation is not available or cannot be instantiated.
258 */
259 public static XMLOutputFactory newFactory(String factoryId,
260     ClassLoader classLoader)
261     throws FactoryConfigurationError {
262     //do not fallback if given classloader can't find the class, throw exception
263     return FactoryFinder.find(XMLOutputFactory.class, factoryId, classLoader, null);
264 }
265
266 /**
267 * Create a new XMLStreamWriter that writes to a writer
268 * @param stream the writer to write to
269 * @throws XMLStreamException
270 */
271 public abstract XMLStreamWriter createXMLStreamWriter(java.io.Writer stream) throws
    XMLStreamException;
272
273 /**
274 * Create a new XMLStreamWriter that writes to a stream
275 * @param stream the stream to write to
276 * @throws XMLStreamException
277 */
278 public abstract XMLStreamWriter createXMLStreamWriter(java.io.OutputStream stream) throws
    XMLStreamException;
279

```

```
280  /**
281   * Create a new XMLStreamWriter that writes to a stream
282   * @param stream the stream to write to
283   * @param encoding the encoding to use
284   * @throws XMLStreamException
285   */
286  public abstract XMLStreamWriter createXMLStreamWriter(java.io.OutputStream stream,
287                                                       String encoding) throws XMLStreamException;
288
289  /**
290   * Create a new XMLStreamWriter that writes to a JAXP result. This method is optional.
291   * @param result the result to write to
292   * @throws UnsupportedOperationException if this method is not
293   * supported by this XMLOutputFactory
294   * @throws XMLStreamException
295   */
296  public abstract XMLStreamWriter createXMLStreamWriter(Result result) throws XMLStreamException;
297
298
299  /**
300   * Create a new XMLEventWriter that writes to a JAXP result. This method is optional.
301   * @param result the result to write to
302   * @throws UnsupportedOperationException if this method is not
303   * supported by this XMLOutputFactory
304   * @throws XMLStreamException
305   */
306  public abstract XMLEventWriter createXMLEventWriter(Result result) throws XMLStreamException;
307
308  /**
309   * Create a new XMLEventWriter that writes to a stream
310   * @param stream the stream to write to
311   * @throws XMLStreamException
312   */
313  public abstract XMLEventWriter createXMLEventWriter(java.io.OutputStream stream) throws
    XMLStreamException;
314
315
316
317  /**
318   * Create a new XMLEventWriter that writes to a stream
319   * @param stream the stream to write to
320   * @param encoding the encoding to use
321   * @throws XMLStreamException
322   */
323  public abstract XMLEventWriter createXMLEventWriter(java.io.OutputStream stream,
324                                                       String encoding) throws XMLStreamException;
325
326  /**
327   * Create a new XMLEventWriter that writes to a writer
328   * @param stream the stream to write to
329   * @throws XMLStreamException
330   */
331  public abstract XMLEventWriter createXMLEventWriter(java.io.Writer stream) throws
```

```

XMLStreamException;
332
333 /**
334  * Allows the user to set specific features/properties on the underlying implementation.
335  * @param name The name of the property
336  * @param value The value of the property
337  * @throws java.lang.IllegalArgumentException if the property is not supported
338  */
339 public abstract void setProperty(java.lang.String name,
340                                Object value)
341     throws IllegalArgumentException;
342
343 /**
344  * Get a feature/property on the underlying implementation
345  * @param name The name of the property
346  * @return The value of the property
347  * @throws java.lang.IllegalArgumentException if the property is not supported
348  */
349 public abstract Object getProperty(java.lang.String name)
350     throws IllegalArgumentException;
351
352 /**
353  * Query the set of properties that this factory supports.
354  *
355  * @param name The name of the property (may not be null)
356  * @return true if the property is supported and false otherwise
357  */
358 public abstract boolean isPropertySupported(String name);
359 }

```

## 18.12 XMLReporter.java

Secuencia 201: javax/xml/stream/XMLReporter.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```

```

21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  /**
32   * This interface is used to report non-fatal errors.
33   * Only warnings should be echoed through this interface.
34   * @version 1.0
35   * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
36   * @since 1.6
37   */
38  public interface XMLReporter {
39
40      /**
41
42       * Report the desired message in an application specific format.
43
44       * Only warnings and non-fatal errors should be reported through
45
46       * this interface.
47
48       * Fatal errors should be thrown as XMLStreamException.
49
50       *
51
52       * @param message the error message
53
54       * @param errorType an implementation defined error type
55
56       * @param relatedInformation information related to the error, if available
57
58       * @param location the location of the error, if available
59
60       * @throws XMLStreamException
61
62       */
63      public void report(String message, String errorType, Object relatedInformation, Location
        location)
        throws XMLStreamException;
64  }
65

```

## 18.13 XMLResolver.java

Secuencia 202: javax/xml/stream/XMLResolver.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29 package javax.xml.stream;
30
31 /**
32  * This interface is used to resolve resources during an XML parse. If an application wishes to
33  * perform custom entity resolution it must register an instance of this interface with
34  * the XMLInputFactory using the setXMLResolver method.
35  *
36  * @version 1.0
37  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
38  * @since 1.6
39  */
40 public interface XMLResolver {
41
42     /**
43      * Retrieves a resource. This resource can be of the following three return types:
44      * (1) java.io.InputStream (2) javax.xml.stream.XMLStreamReader (3) java.xml.stream.
45      *   XMLEventReader.
46      * If this method returns null the processor will attempt to resolve the entity using its
47      * default mechanism.
48      *
49      * @param publicID The public identifier of the external entity being referenced, or null if none
50      *   was supplied.
51      * @param systemID The system identifier of the external entity being referenced.
52      * @param baseURI Absolute base URI associated with systemId.
53      * @param namespace The namespace of the entity to resolve.
54      * @return The resource requested or null.
55      * @throws XMLStreamException if there was a failure attempting to resolve the resource.
56      */
57 }
```



```

55 public Object resolveEntity(String publicID,
56                             String systemID,
57                             String baseURI,
58                             String namespace)
59     throws XMLStreamException;
60 }

```

## 18.14 XMLStreamConstants.java

Secuencia 203: javax/xml/stream/XMLStreamConstants.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 package javax.xml.stream;
26
27 /**
28 * This interface declares the constants used in this API.
29 * Numbers in the range 0 to 256 are reserved for the specification,
30 * user defined events must use event codes outside that range.
31 *
32 * @since 1.6
33 */
34
35 public interface XMLStreamConstants {
36     /**
37      * Indicates an event is a start element
38      * @see javax.xml.stream.events.StartElement
39      */
40     public static final int START_ELEMENT=1;
41     /**
42      * Indicates an event is an end element
43      * @see javax.xml.stream.events.EndElement

```

```
44  */
45  public static final int END_ELEMENT=2;
46  /**
47   * Indicates an event is a processing instruction
48   * @see javax.xml.stream.events.ProcessingInstruction
49   */
50  public static final int PROCESSING_INSTRUCTION=3;
51
52  /**
53   * Indicates an event is characters
54   * @see javax.xml.stream.events.Characters
55   */
56  public static final int CHARACTERS=4;
57
58  /**
59   * Indicates an event is a comment
60   * @see javax.xml.stream.events.Comment
61   */
62  public static final int COMMENT=5;
63
64  /**
65   * The characters are white space
66   * (see [XML], 2.10 "White Space Handling").
67   * Events are only reported as SPACE if they are ignorable white
68   * space. Otherwise they are reported as CHARACTERS.
69   * @see javax.xml.stream.events.Characters
70   */
71  public static final int SPACE=6;
72
73  /**
74   * Indicates an event is a start document
75   * @see javax.xml.stream.events.StartDocument
76   */
77  public static final int START_DOCUMENT=7;
78
79  /**
80   * Indicates an event is an end document
81   * @see javax.xml.stream.events.EndDocument
82   */
83  public static final int END_DOCUMENT=8;
84
85  /**
86   * Indicates an event is an entity reference
87   * @see javax.xml.stream.events.EntityReference
88   */
89  public static final int ENTITY_REFERENCE=9;
90
91  /**
92   * Indicates an event is an attribute
93   * @see javax.xml.stream.events.Attribute
94   */
95  public static final int ATTRIBUTE=10;
96
```

```

97  /**
98   * Indicates an event is a DTD
99   * @see javax.xml.stream.events.DTD
100  */
101  public static final int DTD=11;
102
103  /**
104   * Indicates an event is a CDATA section
105   * @see javax.xml.stream.events.Characters
106  */
107  public static final int CDATA=12;
108
109  /**
110   * Indicates the event is a namespace declaration
111   *
112   * @see javax.xml.stream.events.Namespace
113  */
114  public static final int NAMESPACE=13;
115
116  /**
117   * Indicates a Notation
118   * @see javax.xml.stream.events.NotationDeclaration
119  */
120  public static final int NOTATION_DECLARATION=14;
121
122  /**
123   * Indicates a Entity Declaration
124   * @see javax.xml.stream.events.NotationDeclaration
125  */
126  public static final int ENTITY_DECLARATION=15;
127  }

```

## 18.15 XMLStreamException.java

Secuencia 204: javax/xml/stream/XMLStreamException.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *

```

```
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  /**
32   * The base exception for unexpected processing errors. This Exception
33   * class is used to report well-formedness errors as well as unexpected
34   * processing conditions.
35   * @version 1.0
36   * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
37   * @since 1.6
38   */
39
40  public class XMLStreamException extends Exception {
41
42      protected Throwable nested;
43      protected Location location;
44
45      /**
46       * Default constructor
47       */
48      public XMLStreamException(){
49          super();
50      }
51
52      /**
53       * Construct an exception with the associated message.
54       *
55       * @param msg the message to report
56       */
57      public XMLStreamException(String msg) {
58          super(msg);
59      }
60
61      /**
62       * Construct an exception with the associated exception
63       *
64       * @param th a nested exception
65       */
66      public XMLStreamException(Throwable th) {
67          super(th);
68          nested = th;
69      }
70
71      /**
```

```
72  * Construct an exception with the associated message and exception
73  *
74  * @param th a nested exception
75  * @param msg the message to report
76  */
77  public XMLStreamException(String msg, Throwable th) {
78      super(msg, th);
79      nested = th;
80  }
81
82  /**
83   * Construct an exception with the associated message, exception and location.
84   *
85   * @param th a nested exception
86   * @param msg the message to report
87   * @param location the location of the error
88   */
89  public XMLStreamException(String msg, Location location, Throwable th) {
90      super("ParseError at [row,col]:["+location.getLineNumber()+", "+
91          location.getColumnNumber()+"]\n"+
92          "Message: "+msg);
93      nested = th;
94      this.location = location;
95  }
96
97  /**
98   * Construct an exception with the associated message, exception and location.
99   *
100  * @param msg the message to report
101  * @param location the location of the error
102  */
103  public XMLStreamException(String msg,
104                          Location location) {
105      super("ParseError at [row,col]:["+location.getLineNumber()+", "+
106          location.getColumnNumber()+"]\n"+
107          "Message: "+msg);
108      this.location = location;
109  }
110
111
112  /**
113   * Gets the nested exception.
114   *
115   * @return Nested exception
116   */
117  public Throwable getNestedException() {
118      return nested;
119  }
120
121  /**
122   * Gets the location of the exception
123   *
124   * @return the location of the exception, may be null if none is available
```

```
125  */
126  public Location getLocation() {
127      return location;
128  }
129
130 }
```

## 18.16 XMLStreamReader.java

Secuencia 205: javax/xml/stream/XMLStreamReader.java

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26   * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27   */
28
29  package javax.xml.stream;
30
31  import java.io.Reader;
32  import javax.xml.namespace.NamespaceContext;
33  import javax.xml.namespace.QName;
34
35  /**
36   * The XMLStreamReader interface allows forward, read-only access to XML.
37   * It is designed to be the lowest level and most efficient way to
38   * read XML data.
39   *
40   * <p> The XMLStreamReader is designed to iterate over XML using
41   * next() and hasNext(). The data can be accessed using methods such as getEventType(),
42   * getNamespaceURI(), getLocalName() and getText();
43   *
```

```

44 * <p> The <a href="#next()">next()</a> method causes the reader to read the next parse event.
45 * The next() method returns an integer which identifies the type of event just read.
46 * <p> The event type can be determined using <a href="#getEventType()">getEventType()</a>.
47 * <p> Parsing events are defined as the XML Declaration, a DTD,
48 * start tag, character data, white space, end tag, comment,
49 * or processing instruction. An attribute or namespace event may be encountered
50 * at the root level of a document as the result of a query operation.
51 *
52 * <p>For XML 1.0 compliance an XML processor must pass the
53 * identifiers of declared unparsed entities, notation declarations and their
54 * associated identifiers to the application. This information is
55 * provided through the property API on this interface.
56 * The following two properties allow access to this information:
57 * javax.xml.stream.notation and javax.xml.stream.entities.
58 * When the current event is a DTD the following call will return a
59 * list of Notations
60 * <code>List l = (List) getProperty("javax.xml.stream.notation");</code>
61 * The following call will return a list of entity declarations:
62 * <code>List l = (List) getProperty("javax.xml.stream.entities");</code>
63 * These properties can only be accessed during a DTD event and
64 * are defined to return null if the information is not available.
65 *
66 * <p>The following table describes which methods are valid in what state.
67 * If a method is called in an invalid state the method will throw a
68 * java.lang.IllegalStateException.
69 *
70 * <table border="2" rules="all" cellpadding="4">
71 *   <thead>
72 *     <tr>
73 *       <th align="center" colspan="2">
74 *         Valid methods for each state
75 *       </th>
76 *     </tr>
77 *   </thead>
78 *   <tbody>
79 *     <tr>
80 *       <th>Event Type</th>
81 *       <th>Valid Methods</th>
82 *     </tr>
83 *     <tr>
84 *       <td> All States </td>
85 *       <td> getProperty(), hasNext(), require(), close(),
86 *         getNamespaceURI(), isStartElement(),
87 *         isEndElement(), isCharacters(), isWhiteSpace(),
88 *         getNamespaceContext(), getEventType(), getLocation(),
89 *         hasText(), hasName()
90 *       </td>
91 *     </tr>
92 *     <tr>
93 *       <td> START_ELEMENT </td>
94 *       <td> next(), getName(), getLocalName(), hasName(), getPrefix(),
95 *         getAttributeXXX(), isAttributeSpecified(),
96 *         getNamespaceXXX(),

```

```

97 *         getElementText(), nextTag()
98 *     </td>
99 * </tr>
100 *     <td> ATTRIBUTE </td>
101 *     <td> next(), nextTag()
102 *         getAttributeXXX(), isAttributeSpecified(),
103 *     </td>
104 * </tr>
105 * </tr>
106 *     <td> NAMESPACE </td>
107 *     <td> next(), nextTag()
108 *         getNamespaceXXX()
109 *     </td>
110 * </tr>
111 * <tr>
112 *     <td> END_ELEMENT </td>
113 *     <td> next(), getName(), getLocalName(), hasName(), getPrefix(),
114 *         getNamespaceXXX(), nextTag()
115 *     </td>
116 * </tr>
117 * <tr>
118 *     <td> CHARACTERS </td>
119 *     <td> next(), getTextXXX(), nextTag() </td>
120 * </tr>
121 * <tr>
122 *     <td> CDATA </td>
123 *     <td> next(), getTextXXX(), nextTag() </td>
124 * </tr>
125 * <tr>
126 *     <td> COMMENT </td>
127 *     <td> next(), getTextXXX(), nextTag() </td>
128 * </tr>
129 * <tr>
130 *     <td> SPACE </td>
131 *     <td> next(), getTextXXX(), nextTag() </td>
132 * </tr>
133 * <tr>
134 *     <td> START_DOCUMENT </td>
135 *     <td> next(), getEncoding(), getVersion(), isStandalone(), standaloneSet(),
136 *         getCharacterEncodingScheme(), nextTag()</td>
137 * </tr>
138 * <tr>
139 *     <td> END_DOCUMENT </td>
140 *     <td> close()</td>
141 * </tr>
142 * <tr>
143 *     <td> PROCESSING_INSTRUCTION </td>
144 *     <td> next(), getPITarget(), getPIData(), nextTag() </td>
145 * </tr>
146 * <tr>
147 *     <td> ENTITY_REFERENCE </td>
148 *     <td> next(), getLocalName(), getText(), nextTag() </td>
149 * </tr>

```



```

150 *      <tr>
151 *          <td> DTD </td>
152 *          <td> next(), getText(), nextTag() </td>
153 *      </tr>
154 *  </tbody>
155 *  </table>
156 *
157 * @version 1.0
158 * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
159 * @see javax.xml.stream.events.XMLEvent
160 * @see XMLInputFactory
161 * @see XMLStreamWriter
162 * @since 1.6
163 */
164 public interface XMLStreamReader extends XMLStreamConstants {
165     /**
166      * Get the value of a feature/property from the underlying implementation
167      * @param name The name of the property, may not be null
168      * @return The value of the property
169      * @throws IllegalArgumentException if name is null
170      */
171     public Object getProperty(java.lang.String name) throws java.lang.IllegalArgumentException;
172
173     /**
174      * Get next parsing event - a processor may return all contiguous
175      * character data in a single chunk, or it may split it into several chunks.
176      * If the property javax.xml.stream.isCoalescing is set to true
177      * element content must be coalesced and only one CHARACTERS event
178      * must be returned for contiguous element content or
179      * CDATA Sections.
180      *
181      * By default entity references must be
182      * expanded and reported transparently to the application.
183      * An exception will be thrown if an entity reference cannot be expanded.
184      * If element content is empty (i.e. content is "") then no CHARACTERS event will be reported.
185      *
186      * <p>Given the following XML:<br>
187      * <foo><!--description-->content text<![CDATA[<greeting>Hello</greeting>]]>other
188      *   content</foo><br>
189      * The behavior of calling next() when being on foo will be:<br>
190      * 1- the comment (COMMENT)<br>
191      * 2- then the characters section (CHARACTERS)<br>
192      * 3- then the CDATA section (another CHARACTERS)<br>
193      * 4- then the next characters section (another CHARACTERS)<br>
194      * 5- then the END_ELEMENT<br>
195      *
196      * <p><b>NOTE:</b> empty element (such as <tag/>) will be reported
197      *   with two separate events: START_ELEMENT, END_ELEMENT - This preserves
198      *   parsing equivalency of empty element to <tag></tag>.
199      *
200      * This method will throw an IllegalStateException if it is called after hasNext() returns false.
201      * @see javax.xml.stream.events.XMLEvent
202      * @return the integer code corresponding to the current parse event

```

```

202  * @throws NoSuchElementException if this is called when hasNext() returns false
203  * @throws XMLStreamException if there is an error processing the underlying XML source
204  */
205  public int next() throws XMLStreamException;
206
207  /**
208   * Test if the current event is of the given type and if the namespace and name match the current
209   * namespace and name of the current event. If the namespaceURI is null it is not checked for
210   * equality,
211   * if the localName is null it is not checked for equality.
212   * @param type the event type
213   * @param namespaceURI the uri of the event, may be null
214   * @param localName the localName of the event, may be null
215   * @throws XMLStreamException if the required values are not matched.
216   */
217  public void require(int type, String namespaceURI, String localName) throws XMLStreamException;
218
219  /**
220   * Reads the content of a text-only element, an exception is thrown if this is
221   * not a text-only element.
222   * Regardless of value of javax.xml.stream.isCoalescing this method always returns coalesced
223   * content.
224   * <br /> Precondition: the current event is START_ELEMENT.
225   * <br /> Postcondition: the current event is the corresponding END_ELEMENT.
226   *
227   * <br />The method does the following (implementations are free to optimized
228   * but must do equivalent processing):
229   * <pre>
230   * if(getEventType() != XMLStreamConstants.START_ELEMENT) {
231   *   throw new XMLStreamException(
232   *     "parser must be on START_ELEMENT to read next text", getLocation());
233   * }
234   * int eventType = next();
235   * StringBuffer content = new StringBuffer();
236   * while(eventType != XMLStreamConstants.END_ELEMENT ) {
237   *   if(eventType == XMLStreamConstants.CHARACTERS
238   *   || eventType == XMLStreamConstants.CDATA
239   *   || eventType == XMLStreamConstants.SPACE
240   *   || eventType == XMLStreamConstants.ENTITY_REFERENCE) {
241   *     buf.append(getText());
242   *   } else if(eventType == XMLStreamConstants.PROCESSING_INSTRUCTION
243   *   || eventType == XMLStreamConstants.COMMENT) {
244   *     // skipping
245   *   } else if(eventType == XMLStreamConstants.END_DOCUMENT) {
246   *     throw new XMLStreamException(
247   *       "unexpected end of document when reading element text content", this);
248   *   } else if(eventType == XMLStreamConstants.START_ELEMENT) {
249   *     throw new XMLStreamException(
250   *       "element text content may not contain START_ELEMENT", getLocation());
251   *   } else {
252   *     throw new XMLStreamException(
253   *       "Unexpected event type "+eventType, getLocation());
254   *   }

```

```

253     * eventType = next();
254     * }
255     * return buf.toString();
256     * </pre>
257     *
258     * @throws XMLStreamException if the current event is not a START_ELEMENT
259     * or if a non text element is encountered
260     */
261     public String getElementText() throws XMLStreamException;
262
263     /**
264     * Skips any white space (isWhiteSpace() returns true), COMMENT,
265     * or PROCESSING_INSTRUCTION,
266     * until a START_ELEMENT or END_ELEMENT is reached.
267     * If other than white space characters, COMMENT, PROCESSING_INSTRUCTION, START_ELEMENT,
268     * END_ELEMENT
269     * are encountered, an exception is thrown. This method should
270     * be used when processing element-only content separated by white space.
271     *
272     * <br /> Precondition: none
273     * <br /> Postcondition: the current event is START_ELEMENT or END_ELEMENT
274     * and cursor may have moved over any whitespace event.
275     *
276     * <br /> Essentially it does the following (implementations are free to optimized
277     * but must do equivalent processing):
278     * <pre>
279     * int eventType = next();
280     * while((eventType == XMLStreamConstants.CHARACTERS && isWhiteSpace()) // skip
281     *     whitespace
282     * || (eventType == XMLStreamConstants.CDATA && isWhiteSpace())
283     * // skip whitespace
284     * || eventType == XMLStreamConstants.SPACE
285     * || eventType == XMLStreamConstants.PROCESSING_INSTRUCTION
286     * || eventType == XMLStreamConstants.COMMENT
287     * ) {
288     *     eventType = next();
289     * }
290     * if (eventType != XMLStreamConstants.START_ELEMENT && eventType != XMLStreamConstants.
291     *     END_ELEMENT) {
292     *     throw new String XMLStreamException("expected start or end tag", getLocation());
293     * }
294     * return eventType;
295     * </pre>
296     *
297     * @return the event type of the element read (START_ELEMENT or END_ELEMENT)
298     * @throws XMLStreamException if the current event is not white space, PROCESSING_INSTRUCTION,
299     * START_ELEMENT or END_ELEMENT
300     * @throws NoSuchElementException if this is called when hasNext() returns false
301     */
302     public int nextTag() throws XMLStreamException;
303
304     /**
305     * Returns true if there are more parsing events and false

```

```
303     * if there are no more events. This method will return
304     * false if the current state of the XMLStreamReader is
305     * END_DOCUMENT
306     * @return true if there are more events, false otherwise
307     * @throws XMLStreamException if there is a fatal error detecting the next state
308     */
309     public boolean hasNext() throws XMLStreamException;
310
311     /**
312     * Frees any resources associated with this Reader. This method does not close the
313     * underlying input source.
314     * @throws XMLStreamException if there are errors freeing associated resources
315     */
316     public void close() throws XMLStreamException;
317
318     /**
319     * Return the uri for the given prefix.
320     * The uri returned depends on the current state of the processor.
321     *
322     * <p><strong>NOTE:</strong>The 'xml' prefix is bound as defined in
323     * <a href="http://www.w3.org/TR/REC-xml-names/#ns-using">Namespaces in XML</a>
324     * specification to "http://www.w3.org/XML/1998/namespace".
325     *
326     * <p><strong>NOTE:</strong> The 'xmlns' prefix must be resolved to following namespace
327     * <a href="http://www.w3.org/2000/xmlns/">http://www.w3.org/2000/xmlns/</a>
328     * @param prefix The prefix to lookup, may not be null
329     * @return the uri bound to the given prefix or null if it is not bound
330     * @throws IllegalArgumentException if the prefix is null
331     */
332     public String getNamespaceURI(String prefix);
333
334     /**
335     * Returns true if the cursor points to a start tag (otherwise false)
336     * @return true if the cursor points to a start tag, false otherwise
337     */
338     public boolean isStartElement();
339
340     /**
341     * Returns true if the cursor points to an end tag (otherwise false)
342     * @return true if the cursor points to an end tag, false otherwise
343     */
344     public boolean isEndElement();
345
346     /**
347     * Returns true if the cursor points to a character data event
348     * @return true if the cursor points to character data, false otherwise
349     */
350     public boolean isCharacters();
351
352     /**
353     * Returns true if the cursor points to a character data event
354     * that consists of all whitespace
355     * @return true if the cursor points to all whitespace, false otherwise
```

```
356     */
357     public boolean isWhiteSpace();
358
359
360     /**
361      * Returns the normalized attribute value of the
362      * attribute with the namespace and localName
363      * If the namespaceURI is null the namespace
364      * is not checked for equality
365      * @param namespaceURI the namespace of the attribute
366      * @param localName the local name of the attribute, cannot be null
367      * @return returns the value of the attribute , returns null if not found
368      * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
369      */
370     public String getAttributeValue(String namespaceURI,
371                                     String localName);
372
373     /**
374      * Returns the count of attributes on this START_ELEMENT,
375      * this method is only valid on a START_ELEMENT or ATTRIBUTE. This
376      * count excludes namespace definitions. Attribute indices are
377      * zero-based.
378      * @return returns the number of attributes
379      * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
380      */
381     public int getAttributeCount();
382
383     /** Returns the qname of the attribute at the provided index
384      *
385      * @param index the position of the attribute
386      * @return the QName of the attribute
387      * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
388      */
389     public QName getAttributeName(int index);
390
391     /**
392      * Returns the namespace of the attribute at the provided
393      * index
394      * @param index the position of the attribute
395      * @return the namespace URI (can be null)
396      * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
397      */
398     public String getAttributeNamespace(int index);
399
400     /**
401      * Returns the localName of the attribute at the provided
402      * index
403      * @param index the position of the attribute
404      * @return the localName of the attribute
405      * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
406      */
407     public String getAttributeLocalName(int index);
408
```

```
409  /**
410   * Returns the prefix of this attribute at the
411   * provided index
412   * @param index the position of the attribute
413   * @return the prefix of the attribute
414   * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
415   */
416  public String getAttributePrefix(int index);
417
418  /**
419   * Returns the XML type of the attribute at the provided
420   * index
421   * @param index the position of the attribute
422   * @return the XML type of the attribute
423   * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
424   */
425  public String getAttributeType(int index);
426
427  /**
428   * Returns the value of the attribute at the
429   * index
430   * @param index the position of the attribute
431   * @return the attribute value
432   * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
433   */
434  public String getAttributeValue(int index);
435
436  /**
437   * Returns a boolean which indicates if this
438   * attribute was created by default
439   * @param index the position of the attribute
440   * @return true if this is a default attribute
441   * @throws IllegalStateException if this is not a START_ELEMENT or ATTRIBUTE
442   */
443  public boolean isAttributeSpecified(int index);
444
445  /**
446   * Returns the count of namespaces declared on this START_ELEMENT or END_ELEMENT,
447   * this method is only valid on a START_ELEMENT, END_ELEMENT or NAMESPACE. On
448   * an END_ELEMENT the count is of the namespaces that are about to go
449   * out of scope. This is the equivalent of the information reported
450   * by SAX callback for an end element event.
451   * @return returns the number of namespace declarations on this specific element
452   * @throws IllegalStateException if this is not a START_ELEMENT, END_ELEMENT or NAMESPACE
453   */
454  public int getNamespaceCount();
455
456  /**
457   * Returns the prefix for the namespace declared at the
458   * index. Returns null if this is the default namespace
459   * declaration
460   *
461   * @param index the position of the namespace declaration
```

```
462     * @return returns the namespace prefix
463     * @throws IllegalStateException if this is not a START_ELEMENT, END_ELEMENT or NAMESPACE
464     */
465     public String getNamespacePrefix(int index);
466
467     /**
468     * Returns the uri for the namespace declared at the
469     * index.
470     *
471     * @param index the position of the namespace declaration
472     * @return returns the namespace uri
473     * @throws IllegalStateException if this is not a START_ELEMENT, END_ELEMENT or NAMESPACE
474     */
475     public String getNamespaceURI(int index);
476
477     /**
478     * Returns a read only namespace context for the current
479     * position. The context is transient and only valid until
480     * a call to next() changes the state of the reader.
481     * @return return a namespace context
482     */
483     public NamespaceContext getNamespaceContext();
484
485     /**
486     * Returns a reader that points to the current start element
487     * and all of its contents. Throws an XMLStreamException if the
488     * cursor does not point to a START_ELEMENT.<p>
489     * The sub stream is read from it MUST be read before the parent stream is
490     * moved on, if not any call on the sub stream will cause an XMLStreamException to be
491     * thrown. The parent stream will always return the same result from next()
492     * whatever is done to the sub stream.
493     * @return an XMLStreamReader which points to the next element
494     */
495     // public XMLStreamReader subReader() throws XMLStreamException;
496
497     /**
498     * Allows the implementation to reset and reuse any underlying tables
499     */
500     // public void recycle() throws XMLStreamException;
501
502     /**
503     * Returns an integer code that indicates the type
504     * of the event the cursor is pointing to.
505     */
506     public int getEventType();
507
508     /**
509     * Returns the current value of the parse event as a string,
510     * this returns the string value of a CHARACTERS event,
511     * returns the value of a COMMENT, the replacement value
512     * for an ENTITY_REFERENCE, the string value of a CDATA section,
513     * the string value for a SPACE event,
514     * or the String value of the internal subset of the DTD.
```

```

515     * If an ENTITY_REFERENCE has been resolved, any character data
516     * will be reported as CHARACTERS events.
517     * @return the current text or null
518     * @throws java.lang.IllegalStateException if this state is not
519     * a valid text state.
520     */
521     public String getText();
522
523     /**
524     * Returns an array which contains the characters from this event.
525     * This array should be treated as read-only and transient. I.e. the array will
526     * contain the text characters until the XMLStreamReader moves on to the next event.
527     * Attempts to hold onto the character array beyond that time or modify the
528     * contents of the array are breaches of the contract for this interface.
529     * @return the current text or an empty array
530     * @throws java.lang.IllegalStateException if this state is not
531     * a valid text state.
532     */
533     public char[] getTextCharacters();
534
535     /**
536     * Gets the the text associated with a CHARACTERS, SPACE or CDATA event.
537     * Text starting a "sourceStart" is copied into "target" starting at "targetStart".
538     * Up to "length" characters are copied. The number of characters actually copied is returned.
539     *
540     * The "sourceStart" argument must be greater or equal to 0 and less than or equal to
541     * the number of characters associated with the event. Usually, one requests text starting at a
542     * "sourceStart" of 0.
543     * If the number of characters actually copied is less than the "length", then there is no more
544     * text.
545     * Otherwise, subsequent calls need to be made until all text has been retrieved. For example:
546     *
547     * <code>
548     * int length = 1024;
549     * char[] myBuffer = new char[ length ];
550     * for ( int sourceStart = 0 ; ; sourceStart += length )
551     * {
552     *     int nCopied = stream.getTextCharacters( sourceStart, myBuffer, 0, length );
553     *     if (nCopied < length)
554     *         break;
555     * }
556     * </code>
557     * XMLStreamException may be thrown if there are any XML errors in the underlying source.
558     * The "targetStart" argument must be greater than or equal to 0 and less than the length of "
559     * target",
560     * Length must be greater than 0 and "targetStart + length" must be less than or equal to length
561     * of "target".
562     *
563     * @param sourceStart the index of the first character in the source array to copy
564     * @param target the destination array
565     * @param targetStart the start offset in the target array

```



```
564 * @param length the number of characters to copy
565 * @return the number of characters actually copied
566 * @throws XMLStreamException if the underlying XML source is not well-formed
567 * @throws IndexOutOfBoundsException if targetStart < 0 or > than the length of target
568 * @throws IndexOutOfBoundsException if length < 0 or targetStart + length > length of target
569 * @throws UnsupportedOperationException if this method is not supported
570 * @throws NullPointerException is if target is null
571 */
572 public int getTextCharacters(int sourceStart, char[] target, int targetStart, int length)
573     throws XMLStreamException;
574
575 /**
576 * Gets the text associated with a CHARACTERS, SPACE or CDATA event. Allows the underlying
577 * implementation to return the text as a stream of characters. The reference to the
578 * Reader returned by this method is only valid until next() is called.
579 *
580 * All characters must have been checked for well-formedness.
581 *
582 * <p> This method is optional and will throw UnsupportedOperationException if it is not
583     supported.
584 * @throws UnsupportedOperationException if this method is not supported
585 * @throws IllegalStateException if this is not a valid text state
586 */
587 //public Reader getTextStream();
588
589 /**
590 * Returns the offset into the text character array where the first
591 * character (of this text event) is stored.
592 * @throws java.lang.IllegalStateException if this state is not
593 * a valid text state.
594 */
595 public int getTextStart();
596
597 /**
598 * Returns the length of the sequence of characters for this
599 * Text event within the text character array.
600 * @throws java.lang.IllegalStateException if this state is not
601 * a valid text state.
602 */
603 public int getTextLength();
604
605 /**
606 * Return input encoding if known or null if unknown.
607 * @return the encoding of this instance or null
608 */
609 public String getEncoding();
610
611 /**
612 * Return true if the current event has text, false otherwise
613 * The following events have text:
614 * CHARACTERS,DTD ,ENTITY_REFERENCE, COMMENT, SPACE
615 */
616 public boolean hasText();
```

```
616
617 /**
618  * Return the current location of the processor.
619  * If the Location is unknown the processor should return
620  * an implementation of Location that returns -1 for the
621  * location and null for the publicId and systemId.
622  * The location information is only valid until next() is
623  * called.
624  */
625 public Location getLocation();
626
627 /**
628  * Returns a QName for the current START_ELEMENT or END_ELEMENT event
629  * @return the QName for the current START_ELEMENT or END_ELEMENT event
630  * @throws IllegalStateException if this is not a START_ELEMENT or
631  * END_ELEMENT
632  */
633 public QName getName();
634
635 /**
636  * Returns the (local) name of the current event.
637  * For START_ELEMENT or END_ELEMENT returns the (local) name of the current element.
638  * For ENTITY_REFERENCE it returns entity name.
639  * The current event must be START_ELEMENT or END_ELEMENT,
640  * or ENTITY_REFERENCE
641  * @return the localName
642  * @throws IllegalStateException if this not a START_ELEMENT,
643  * END_ELEMENT or ENTITY_REFERENCE
644  */
645 public String getLocalName();
646
647 /**
648  * returns true if the current event has a name (is a START_ELEMENT or END_ELEMENT)
649  * returns false otherwise
650  */
651 public boolean hasName();
652
653 /**
654  * If the current event is a START_ELEMENT or END_ELEMENT this method
655  * returns the URI of the prefix or the default namespace.
656  * Returns null if the event does not have a prefix.
657  * @return the URI bound to this elements prefix, the default namespace, or null
658  */
659 public String getNamespaceURI();
660
661 /**
662  * Returns the prefix of the current event or null if the event does not have a prefix
663  * @return the prefix or null
664  */
665 public String getPrefix();
666
667 /**
668  * Get the xml version declared on the xml declaration
```

```

669     * Returns null if none was declared
670     * @return the XML version or null
671     */
672     public String getVersion();
673
674     /**
675     * Get the standalone declaration from the xml declaration
676     * @return true if this is standalone, or false otherwise
677     */
678     public boolean isStandalone();
679
680     /**
681     * Checks if standalone was set in the document
682     * @return true if standalone was set in the document, or false otherwise
683     */
684     public boolean standaloneSet();
685
686     /**
687     * Returns the character encoding declared on the xml declaration
688     * Returns null if none was declared
689     * @return the encoding declared in the document or null
690     */
691     public String getCharacterEncodingScheme();
692
693     /**
694     * Get the target of a processing instruction
695     * @return the target or null
696     */
697     public String getPITarget();
698
699     /**
700     * Get the data section of a processing instruction
701     * @return the data or null
702     */
703     public String getPIData();
704 }

```

## 18.17 XMLStreamWriter.java

Secuencia 206: javax/xml/stream/XMLStreamWriter.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```

14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream;
30
31  import javax.xml.namespace.NamespaceContext;
32
33  /**
34   * The XMLStreamWriter interface specifies how to write XML. The XMLStreamWriter does
35   * not perform well formedness checking on its input. However
36   * the writeCharacters method is required to escape & , < and >;
37   * For attribute values the writeAttribute method will escape the
38   * above characters plus " to ensure that all character content
39   * and attribute values are well formed.
40   *
41   * Each NAMESPACE
42   * and ATTRIBUTE must be individually written.
43   *
44   * <table border="1" cellpadding="2" cellspacing="0">
45   *   <thead>
46   *     <tr>
47   *       <th colspan="5">XML Namespaces, <code>javax.xml.stream.isRepairingNamespaces</code>
and write method behaviour</th>
48   *     </tr>
49   *     <tr>
50   *       <th>Method</th> <!-- method -->
51   *       <th colspan="2"><code>isRepairingNamespaces</code> == true</th>
52   *       <th colspan="2"><code>isRepairingNamespaces</code> == false</th>
53   *     </tr>
54   *     <tr>
55   *       <th></th> <!-- method -->
56   *       <th>namespaceURI bound</th>
57   *       <th>namespaceURI unbound</th>
58   *       <th>namespaceURI bound</th>
59   *       <th>namespaceURI unbound</th>
60   *     </tr>
61   *   </thead>
62   *
63   *   <tbody>
64   *     <tr>
65   *       <th><code>writeAttribute(namespaceURI, localName, value)</code></th>

```

```

66 *      <!-- isRepairingNamespaces == true -->
67 *      <td>
68 *          <!-- namespaceURI bound -->
69 *          prefix:localName="value"&nbsp;<sup>[1]</sup>
70 *      </td>
71 *      <td>
72 *          <!-- namespaceURI unbound -->
73 *          xmlns:{generated}="namespaceURI" {generated}:localName="value"
74 *      </td>
75 *      <!-- isRepairingNamespaces == false -->
76 *      <td>
77 *          <!-- namespaceURI bound -->
78 *          prefix:localName="value"&nbsp;<sup>[1]</sup>
79 *      </td>
80 *      <td>
81 *          <!-- namespaceURI unbound -->
82 *          <code>XMLStreamException</code>
83 *      </td>
84 *  </tr>
85 *
86 *  <tr>
87 *      <th><code>writeAttribute(prefix, namespaceURI, localName, value)</code></th>
88 *      <!-- isRepairingNamespaces == true -->
89 *      <td>
90 *          <!-- namespaceURI bound -->
91 *          bound to same prefix:<br />
92 *          prefix:localName="value"&nbsp;<sup>[1]</sup><br />
93 *          <br />
94 *          bound to different prefix:<br />
95 *          xmlns:{generated}="namespaceURI" {generated}:localName="value"
96 *      </td>
97 *      <td>
98 *          <!-- namespaceURI unbound -->
99 *          xmlns:prefix="namespaceURI" prefix:localName="value"&nbsp;<sup>[3]</sup>
100 *      </td>
101 *      <!-- isRepairingNamespaces == false -->
102 *      <td>
103 *          <!-- namespaceURI bound -->
104 *          bound to same prefix:<br />
105 *          prefix:localName="value"&nbsp;<sup>[1] [2]</sup><br />
106 *          <br />
107 *          bound to different prefix:<br />
108 *          <code>XMLStreamException</code><sup>[2]</sup>
109 *      </td>
110 *      <td>
111 *          <!-- namespaceURI unbound -->
112 *          xmlns:prefix="namespaceURI" prefix:localName="value"&nbsp;<sup>[2] [5]</sup>
113 *      </td>
114 *  </tr>
115 *
116 *  <tr>
117 *      <th><code>writeStartElement(namespaceURI, localName)</code><br />
118 *      <br />

```

```

119 *         <code>writeEmptyElement(namespaceURI, localName)</code></th>
120 *         <!-- isRepairingNamespaces == true -->
121 *         <td >
122 *             <!-- namespaceURI bound -->
123 *             &lt;prefix:localName&gt;&nbsp;<sup>[1]</sup>
124 *         </td>
125 *         <td>
126 *             <!-- namespaceURI unbound -->
127 *             &lt;{generated}:localName xmlns:{generated}="namespaceURI"&gt;
128 *         </td>
129 *         <!-- isRepairingNamespaces == false -->
130 *         <td>
131 *             <!-- namespaceURI bound -->
132 *             &lt;prefix:localName&gt;&nbsp;<sup>[1]</sup>
133 *         </td>
134 *         <td>
135 *             <!-- namespaceURI unbound -->
136 *             <code>XMLStreamException</code>
137 *         </td>
138 *     </tr>
139 *
140 *     <tr>
141 *         <th><code>writeStartElement(prefix, localName, namespaceURI)</code><br />
142 *         <br />
143 *         <code>writeEmptyElement(prefix, localName, namespaceURI)</code></th>
144 *         <!-- isRepairingNamespaces == true -->
145 *         <td>
146 *             <!-- namespaceURI bound -->
147 *             bound to same prefix:<br />
148 *             &lt;prefix:localName&gt;&nbsp;<sup>[1]</sup><br />
149 *             <br />
150 *             bound to different prefix:<br />
151 *             &lt;{generated}:localName xmlns:{generated}="namespaceURI"&gt;
152 *         </td>
153 *         <td>
154 *             <!-- namespaceURI unbound -->
155 *             &lt;prefix:localName xmlns:prefix="namespaceURI"&gt;&nbsp;<sup>[4]</sup>
156 *         </td>
157 *         <!-- isRepairingNamespaces == false -->
158 *         <td>
159 *             <!-- namespaceURI bound -->
160 *             bound to same prefix:<br />
161 *             &lt;prefix:localName&gt;&nbsp;<sup>[1]</sup><br />
162 *             <br />
163 *             bound to different prefix:<br />
164 *             <code>XMLStreamException</code>
165 *         </td>
166 *         <td>
167 *             <!-- namespaceURI unbound -->
168 *             &lt;prefix:localName&gt;&nbsp;<sup>[1]</sup>
169 *         </td>
170 *     </tr>
171 * </tbody>

```

```

172 *      <tfoot>
173 *          <tr>
174 *              <td colspan="5">
175 *                  Notes:
176 *                  <ul>
177 *                      <li>[1] if namespaceURI == default Namespace URI, then no prefix is written
178 *                      <li>[2] if prefix == "" || null && namespaceURI == "", then no prefix or
179 *                          Namespace declaration is generated or written</li>
180 *                      <li>[3] if prefix == "" || null, then a prefix is randomly generated</li>
181 *                      <li>[4] if prefix == "" || null, then it is treated as the default Namespace
182 *                          and no prefix is generated or written, an xmlns declaration is generated and written if the
183 *                          namespaceURI is unbound</li>
184 *                      <li>[5] if prefix == "" || null, then it is treated as an invalid attempt to
185 *                          define the default Namespace and an XMLStreamException is thrown</li>
186 *                  </ul>
187 *              </td>
188 *          </tr>
189 *      </tfoot>
190 *  </table>
191 *
192 * @version 1.0
193 * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
194 * @see XMLOutputFactory
195 * @see XMLStreamReader
196 * @since 1.6
197 */
198 public interface XMLStreamWriter {
199
200     /**
201      * Writes a start tag to the output. All writeStartElement methods
202      * open a new scope in the internal namespace context. Writing the
203      * corresponding EndElement causes the scope to be closed.
204      * @param localName local name of the tag, may not be null
205      * @throws XMLStreamException
206      */
207     public void writeStartElement(String localName)
208         throws XMLStreamException;
209
210     /**
211      * Writes a start tag to the output
212      * @param namespaceURI the namespaceURI of the prefix to use, may not be null
213      * @param localName local name of the tag, may not be null
214      * @throws XMLStreamException if the namespace URI has not been bound to a prefix and
215      *     javax.xml.stream.isRepairingNamespaces has not been set to true
216      */
217     public void writeStartElement(String namespaceURI, String localName)
218         throws XMLStreamException;
219
220     /**
221      * Writes a start tag to the output
222      * @param localName local name of the tag, may not be null
223      * @param prefix the prefix of the tag, may not be null

```

```
220     * @param namespaceURI the uri to bind the prefix to, may not be null
221     * @throws XMLStreamException
222     */
223     public void writeStartElement(String prefix,
224                                   String localName,
225                                   String namespaceURI)
226         throws XMLStreamException;
227
228     /**
229     * Writes an empty element tag to the output
230     * @param namespaceURI the uri to bind the tag to, may not be null
231     * @param localName local name of the tag, may not be null
232     * @throws XMLStreamException if the namespace URI has not been bound to a prefix and
233     * javax.xml.stream.isRepairingNamespaces has not been set to true
234     */
235     public void writeEmptyElement(String namespaceURI, String localName)
236         throws XMLStreamException;
237
238     /**
239     * Writes an empty element tag to the output
240     * @param prefix the prefix of the tag, may not be null
241     * @param localName local name of the tag, may not be null
242     * @param namespaceURI the uri to bind the tag to, may not be null
243     * @throws XMLStreamException
244     */
245     public void writeEmptyElement(String prefix, String localName, String namespaceURI)
246         throws XMLStreamException;
247
248     /**
249     * Writes an empty element tag to the output
250     * @param localName local name of the tag, may not be null
251     * @throws XMLStreamException
252     */
253     public void writeEmptyElement(String localName)
254         throws XMLStreamException;
255
256     /**
257     * Writes string data to the output without checking for well formedness.
258     * The data is opaque to the XMLStreamWriter, i.e. the characters are written
259     * blindly to the underlying output. If the method cannot be supported
260     * in the current writing context the implementation may throw a
261     * UnsupportedOperationException. For example note that any
262     * namespace declarations, end tags, etc. will be ignored and could
263     * interfere with proper maintenance of the writers internal state.
264     *
265     * @param data the data to write
266     */
267     // public void writeRaw(String data) throws XMLStreamException;
268
269     /**
270     * Writes an end tag to the output relying on the internal
271     * state of the writer to determine the prefix and local name
272     * of the event.
```



```
273     * @throws XMLStreamException
274     */
275     public void writeEndElement()
276         throws XMLStreamException;
277
278     /**
279     * Closes any start tags and writes corresponding end tags.
280     * @throws XMLStreamException
281     */
282     public void writeEndDocument()
283         throws XMLStreamException;
284
285     /**
286     * Close this writer and free any resources associated with the
287     * writer. This must not close the underlying output stream.
288     * @throws XMLStreamException
289     */
290     public void close()
291         throws XMLStreamException;
292
293     /**
294     * Write any cached data to the underlying output mechanism.
295     * @throws XMLStreamException
296     */
297     public void flush()
298         throws XMLStreamException;
299
300     /**
301     * Writes an attribute to the output stream without
302     * a prefix.
303     * @param localName the local name of the attribute
304     * @param value the value of the attribute
305     * @throws IllegalStateException if the current state does not allow Attribute writing
306     * @throws XMLStreamException
307     */
308     public void writeAttribute(String localName, String value)
309         throws XMLStreamException;
310
311     /**
312     * Writes an attribute to the output stream
313     * @param prefix the prefix for this attribute
314     * @param namespaceURI the uri of the prefix for this attribute
315     * @param localName the local name of the attribute
316     * @param value the value of the attribute
317     * @throws IllegalStateException if the current state does not allow Attribute writing
318     * @throws XMLStreamException if the namespace URI has not been bound to a prefix and
319     * @throws javax.xml.stream.isRepairingNamespaces has not been set to true
320     */
321
322     public void writeAttribute(String prefix,
323                               String namespaceURI,
324                               String localName,
325                               String value)
```

```
326     throws XMLStreamException;
327
328     /**
329     * Writes an attribute to the output stream
330     * @param namespaceURI the uri of the prefix for this attribute
331     * @param localName the local name of the attribute
332     * @param value the value of the attribute
333     * @throws IllegalStateException if the current state does not allow Attribute writing
334     * @throws XMLStreamException if the namespace URI has not been bound to a prefix and
335     * javax.xml.stream.isRepairingNamespaces has not been set to true
336     */
337     public void writeAttribute(String namespaceURI,
338                               String localName,
339                               String value)
340         throws XMLStreamException;
341
342     /**
343     * Writes a namespace to the output stream
344     * If the prefix argument to this method is the empty string,
345     * "xmlns", or null this method will delegate to writeDefaultNamespace
346     *
347     * @param prefix the prefix to bind this namespace to
348     * @param namespaceURI the uri to bind the prefix to
349     * @throws IllegalStateException if the current state does not allow Namespace writing
350     * @throws XMLStreamException
351     */
352     public void writeNamespace(String prefix, String namespaceURI)
353         throws XMLStreamException;
354
355     /**
356     * Writes the default namespace to the stream
357     * @param namespaceURI the uri to bind the default namespace to
358     * @throws IllegalStateException if the current state does not allow Namespace writing
359     * @throws XMLStreamException
360     */
361     public void writeDefaultNamespace(String namespaceURI)
362         throws XMLStreamException;
363
364     /**
365     * Writes an xml comment with the data enclosed
366     * @param data the data contained in the comment, may be null
367     * @throws XMLStreamException
368     */
369     public void writeComment(String data)
370         throws XMLStreamException;
371
372     /**
373     * Writes a processing instruction
374     * @param target the target of the processing instruction, may not be null
375     * @throws XMLStreamException
376     */
377     public void writeProcessingInstruction(String target)
378         throws XMLStreamException;
```

```
379
380 /**
381  * Writes a processing instruction
382  * @param target the target of the processing instruction, may not be null
383  * @param data the data contained in the processing instruction, may not be null
384  * @throws XMLStreamException
385  */
386 public void writeProcessingInstruction(String target,
387                                     String data)
388     throws XMLStreamException;
389
390 /**
391  * Writes a CData section
392  * @param data the data contained in the CData Section, may not be null
393  * @throws XMLStreamException
394  */
395 public void writeCData(String data)
396     throws XMLStreamException;
397
398 /**
399  * Write a DTD section. This string represents the entire doctype decl production
400  * from the XML 1.0 specification.
401  *
402  * @param dtd the DTD to be written
403  * @throws XMLStreamException
404  */
405 public void writeDTD(String dtd)
406     throws XMLStreamException;
407
408 /**
409  * Writes an entity reference
410  * @param name the name of the entity
411  * @throws XMLStreamException
412  */
413 public void writeEntityRef(String name)
414     throws XMLStreamException;
415
416 /**
417  * Write the XML Declaration. Defaults the XML version to 1.0, and the encoding to utf-8
418  * @throws XMLStreamException
419  */
420 public void writeStartDocument()
421     throws XMLStreamException;
422
423 /**
424  * Write the XML Declaration. Defaults the XML version to 1.0
425  * @param version version of the xml document
426  * @throws XMLStreamException
427  */
428 public void writeStartDocument(String version)
429     throws XMLStreamException;
430
431 /**
```

```
432 * Write the XML Declaration. Note that the encoding parameter does
433 * not set the actual encoding of the underlying output. That must
434 * be set when the instance of the XMLStreamWriter is created using the
435 * XMLOutputFactory
436 * @param encoding encoding of the xml declaration
437 * @param version version of the xml document
438 * @throws XMLStreamException If given encoding does not match encoding
439 * of the underlying stream
440 */
441 public void writeStartDocument(String encoding,
442                               String version)
443     throws XMLStreamException;
444
445 /**
446 * Write text to the output
447 * @param text the value to write
448 * @throws XMLStreamException
449 */
450 public void writeCharacters(String text)
451     throws XMLStreamException;
452
453 /**
454 * Write text to the output
455 * @param text the value to write
456 * @param start the starting position in the array
457 * @param len the number of characters to write
458 * @throws XMLStreamException
459 */
460 public void writeCharacters(char[] text, int start, int len)
461     throws XMLStreamException;
462
463 /**
464 * Gets the prefix the uri is bound to
465 * @return the prefix or null
466 * @throws XMLStreamException
467 */
468 public String getPrefix(String uri)
469     throws XMLStreamException;
470
471 /**
472 * Sets the prefix the uri is bound to. This prefix is bound
473 * in the scope of the current START_ELEMENT / END_ELEMENT pair.
474 * If this method is called before a START_ELEMENT has been written
475 * the prefix is bound in the root scope.
476 * @param prefix the prefix to bind to the uri, may not be null
477 * @param uri the uri to bind to the prefix, may be null
478 * @throws XMLStreamException
479 */
480 public void setPrefix(String prefix, String uri)
481     throws XMLStreamException;
482
483
484 /**
```

```

485     * Binds a URI to the default namespace
486     * This URI is bound
487     * in the scope of the current START_ELEMENT / END_ELEMENT pair.
488     * If this method is called before a START_ELEMENT has been written
489     * the uri is bound in the root scope.
490     * @param uri the uri to bind to the default namespace, may be null
491     * @throws XMLStreamException
492     */
493     public void setDefaultNamespace(String uri)
494         throws XMLStreamException;
495
496     /**
497     * Sets the current namespace context for prefix and uri bindings.
498     * This context becomes the root namespace context for writing and
499     * will replace the current root namespace context. Subsequent calls
500     * to setPrefix and setDefaultNamespace will bind namespaces using
501     * the context passed to the method as the root context for resolving
502     * namespaces. This method may only be called once at the start of
503     * the document. It does not cause the namespaces to be declared.
504     * If a namespace URI to prefix mapping is found in the namespace
505     * context it is treated as declared and the prefix may be used
506     * by the StreamWriter.
507     * @param context the namespace context to use for this writer, may not be null
508     * @throws XMLStreamException
509     */
510     public void setNamespaceContext(NamespaceContext context)
511         throws XMLStreamException;
512
513     /**
514     * Returns the current namespace context.
515     * @return the current NamespaceContext
516     */
517     public NamespaceContext getNamespaceContext();
518
519     /**
520     * Get the value of a feature/property from the underlying implementation
521     * @param name The name of the property, may not be null
522     * @return The value of the property
523     * @throws IllegalArgumentException if the property is not supported
524     * @throws NullPointerException if the name is null
525     */
526     public Object getProperty(java.lang.String name) throws IllegalArgumentException;
527
528 }

```

## 19 Xml Stream Events (javax.xml.stream.events package)

### 19.1 Attribute.java

Secuencia 207: javax/xml/stream/events/Attribute.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29 package javax.xml.stream.events;
30
31 import javax.xml.namespace.QName;
32
33 /**
34  * An interface that contains information about an attribute. Attributes are reported
35  * as a set of events accessible from a StartElement. Other applications may report
36  * Attributes as first-order events, for example as the results of an XPath expression.
37  *
38  * @version 1.0
39  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
40  * @see StartElement
41  * @since 1.6
42  */
43 public interface Attribute extends XMLEvent {
44
45     /**
46      * Returns the QName for this attribute
47      */
48     QName getName();
49
50     /**
51      * Gets the normalized value of this attribute
52      */
53     public String getValue();
54
55     /**
56      * Gets the type of this attribute, default is
```

```

57  * the String "CDATA"
58  * @return the type as a String, default is "CDATA"
59  */
60  public String getDTDType();
61
62  /**
63   * A flag indicating whether this attribute was actually
64   * specified in the start-tag of its element, or was defaulted from the schema.
65   * @return returns true if this was specified in the start element
66   */
67  public boolean isSpecified();
68
69  }

```

## 19.2 Characters.java

Secuencia 208: javax/xml/stream/events/Characters.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30
31 /**
32 * This describes the interface to Characters events.
33 * All text events get reported as Characters events.
34 * Content, CDATA and whitespace are all reported as
35 * Characters events. IgnorableWhitespace, in most cases,
36 * will be set to false unless an element declaration of element

```

```

37  * content is present for the current element.
38  *
39  * @version 1.0
40  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
41  * @since 1.6
42  */
43  public interface Characters extends XMLEvent {
44      /**
45       * Get the character data of this event
46       */
47      public String getData();
48
49      /**
50       * Returns true if this set of Characters
51       * is all whitespace. Whitespace inside a document
52       * is reported as CHARACTERS. This method allows
53       * checking of CHARACTERS events to see if they
54       * are composed of only whitespace characters
55       */
56      public boolean isWhiteSpace();
57
58      /**
59       * Returns true if this is a CData section. If this
60       * event is CData its event type will be CDATA
61       *
62       * If javax.xml.stream.isCoalescing is set to true CDATA Sections
63       * that are surrounded by non CDATA characters will be reported
64       * as a single Characters event. This method will return false
65       * in this case.
66       */
67      public boolean isCData();
68
69      /**
70       * Return true if this is ignorableWhiteSpace. If
71       * this event is ignorableWhiteSpace its event type will
72       * be SPACE.
73       */
74      public boolean isIgnorableWhiteSpace();
75
76  }

```

### 19.3 Comment.java

Secuencia 209: javax/xml/stream/events/Comment.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *

```



```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream.events;
30
31  /**
32   * An interface for comment events
33   *
34   * @version 1.0
35   * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
36   * @since 1.6
37   */
38  public interface Comment extends XMLEvent {
39
40      /**
41       * Return the string data of the comment, returns empty string if it
42       * does not exist
43       */
44      public String getText();
45  }

```

## 19.4 DTD.java

Secuencia 210: javax/xml/stream/events/DTD.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream.events;
30
31  import java.util.List;
32
33  /**
34   * This is the top level interface for events dealing with DTDs
35   *
36   * @version 1.0
37   * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
38   * @since 1.6
39   */
40  public interface DTD extends XMLEvent {
41
42      /**
43       * Returns the entire Document Type Declaration as a string, including
44       * the internal DTD subset.
45       * This may be null if there is not an internal subset.
46       * If it is not null it must return the entire
47       * Document Type Declaration which matches the doctypedecl
48       * production in the XML 1.0 specification
49       */
50      String getDocumentTypeDeclaration();
51
52      /**
53       * Returns an implementation defined representation of the DTD.
54       * This method may return null if no representation is available.
55       */
56      Object getProcessedDTD();
57
58      /**
59       * Return a List containing the notations declared in the DTD.
60       * This list must contain NotationDeclaration events.
61       * @see NotationDeclaration
62       * @return an unordered list of NotationDeclaration events
63       */
64      List getNotations();
65
66      /**
```

```

67  * Return a List containing the general entities,
68  * both external and internal, declared in the DTD.
69  * This list must contain EntityDeclaration events.
70  * @see EntityDeclaration
71  * @return an unordered list of EntityDeclaration events
72  */
73  List getEntities();
74  }

```

## 19.5 EndDocument.java

Secuencia 211: javax/xml/stream/events/EndDocument.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30
31 /**
32  * A marker interface for the end of the document
33  *
34  * @version 1.0
35  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
36  * @since 1.6
37  */
38 public interface EndDocument extends XMLEvent {
39  /**
40   * No methods are defined in this interface.
41   */

```

42 }

## 19.6 EndElement.java

Secuencia 212: javax/xml/stream/events/EndElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30
31 import java.util.Iterator;
32 import javax.xml.namespace.QName;
33 /**
34 * An interface for the end element event. An EndElement is reported
35 * for each End Tag in the document.
36 *
37 * @version 1.0
38 * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
39 * @see XMLEvent
40 * @since 1.6
41 */
42 public interface EndElement extends XMLEvent {
43
44     /**
45      * Get the name of this event
46      * @return the qualified name of this event
47      */
48     public QName getName();
```

```
49
50 /**
51  * Returns an Iterator of namespaces that have gone out
52  * of scope. Returns an empty iterator if no namespaces have gone
53  * out of scope.
54  * @return an Iterator over Namespace interfaces, or an
55  * empty iterator
56  */
57 public Iterator getNamespaces();
58
59 }
```

## 19.7 EntityDeclaration.java

Secuencia 213: javax/xml/stream/events/EntityDeclaration.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30 /**
31  * An interface for handling Entity Declarations
32  *
33  * This interface is used to record and report unparsed entity declarations.
34  *
35  * @version 1.0
36  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
37  * @since 1.6
38  */
```

```

39 public interface EntityDeclaration extends XMLEvent {
40
41     /**
42      * The entity's public identifier, or null if none was given
43      * @return the public ID for this declaration or null
44      */
45     String getPublicId();
46
47     /**
48      * The entity's system identifier.
49      * @return the system ID for this declaration or null
50      */
51     String getSystemId();
52
53     /**
54      * The entity's name
55      * @return the name, may not be null
56      */
57     String getName();
58
59     /**
60      * The name of the associated notation.
61      * @return the notation name
62      */
63     String getNotationName();
64
65     /**
66      * The replacement text of the entity.
67      * This method will only return non-null
68      * if this is an internal entity.
69      * @return null or the replacement text
70      */
71     String getReplacementText();
72
73     /**
74      * Get the base URI for this reference
75      * or null if this information is not available
76      * @return the base URI or null
77      */
78     String getBaseURI();
79
80 }

```

## 19.8 EntityReference.java

Secuencia 214: javax/xml/stream/events/EntityReference.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *

```

```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29 package javax.xml.stream.events;
30 /**
31  * An interface for handling Entity events.
32  *
33  * This event reports entities that have not been resolved
34  * and reports their replacement text unprocessed (if
35  * available). This event will be reported if javax.xml.stream.isReplacingEntityReferences
36  * is set to false. If javax.xml.stream.isReplacingEntityReferences is set to true
37  * entity references will be resolved transparently.
38  *
39  * Entities are handled in two possible ways:
40  *
41  * (1) If javax.xml.stream.isReplacingEntityReferences is set to true
42  * all entity references are resolved and reported as markup transparently.
43  * (2) If javax.xml.stream.isReplacingEntityReferences is set to false
44  * Entity references are reported as an EntityReference Event.
45  *
46  * @version 1.0
47  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
48  * @since 1.6
49  */
50 public interface EntityReference extends XMLEvent {
51
52     /**
53      * Return the declaration of this entity.
54      */
55     EntityDeclaration getDeclaration();
56
57     /**
58      * The name of the entity
59      * @return the entity's name, may not be null
60      */
61 }
```

```
61     String getName();
62 }
```

## 19.9 Namespace.java

Secuencia 215: javax/xml/stream/events/Namespace.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30
31 import javax.xml.namespace.QName;
32
33 /**
34 * An interface that contains information about a namespace.
35 * Namespaces are accessed from a StartElement.
36 *
37 * @version 1.0
38 * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
39 * @see StartElement
40 * @since 1.6
41 */
42 public interface Namespace extends Attribute {
43
44     /**
45      * Gets the prefix, returns "" if this is a default
46      * namespace declaration.
47      */
48 }
```



```
48 public String getPrefix();
49
50 /**
51  * Gets the uri bound to the prefix of this namespace
52  */
53 public String getNamespaceURI();
54
55 /**
56  * returns true if this attribute declares the default namespace
57  */
58 public boolean isDefaultNamespaceDeclaration();
59 }
```

## 19.10 NotationDeclaration.java

Secuencia 216: javax/xml/stream/events/NotationDeclaration.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30 /**
31  * An interface for handling Notation Declarations
32  *
33  * Receive notification of a notation declaration event.
34  * It is up to the application to record the notation for later reference,
35  * At least one of publicId and systemId must be non-null.
36  * There is no guarantee that the notation declaration
37  * will be reported before any unparsed entities that use it.
```

```
38  *
39  * @version 1.0
40  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
41  * @since 1.6
42  */
43  public interface NotationDeclaration extends XMLEvent {
44      /**
45       * The notation name.
46       */
47      String getName();
48
49      /**
50       * The notation's public identifier, or null if none was given.
51       */
52      String getPublicId();
53
54      /**
55       * The notation's system identifier, or null if none was given.
56       */
57      String getSystemId();
58  }
```

## 19.11 ProcessingInstruction.java

Secuencia 217: javax/xml/stream/events/ProcessingInstruction.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
```

```

29 package javax.xml.stream.events;
30 /**
31  * An interface that describes the data found in processing instructions
32  *
33  * @version 1.0
34  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
35  * @since 1.6
36  */
37 public interface ProcessingInstruction extends XMLEvent {
38
39     /**
40      * The target section of the processing instruction
41      *
42      * @return the String value of the PI or null
43      */
44     public String getTarget();
45
46     /**
47      * The data section of the processing instruction
48      *
49      * @return the String value of the PI's data or null
50      */
51     public String getData();
52 }

```

## 19.12 StartDocument.java

Secuencia 218: javax/xml/stream/events/StartDocument.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*

```

```
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream.events;
30  /**
31   * An interface for the start document event
32   *
33   * @version 1.0
34   * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
35   * @since 1.6
36   */
37  public interface StartDocument extends XMLEvent {
38
39      /**
40       * Returns the system ID of the XML data
41       * @return the system ID, defaults to ""
42       */
43      public String getSystemId();
44
45      /**
46       * Returns the encoding style of the XML data
47       * @return the character encoding, defaults to "UTF-8"
48       */
49      public String getCharacterEncodingScheme();
50
51      /**
52       * Returns true if CharacterEncodingScheme was set in
53       * the encoding declaration of the document
54       */
55      public boolean encodingSet();
56
57      /**
58       * Returns if this XML is standalone
59       * @return the standalone state of XML, defaults to "no"
60       */
61      public boolean isStandalone();
62
63      /**
64       * Returns true if the standalone attribute was set in
65       * the encoding declaration of the document.
66       */
67      public boolean standaloneSet();
68
69      /**
70       * Returns the version of XML of this XML stream
71       * @return the version of XML, defaults to "1.0"
72       */
73      public String getVersion();
74  }
```

## 19.13 StartElement.java

Secuencia 219: javax/xml/stream/events/StartElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30
31 import javax.xml.namespace.QName;
32 import javax.xml.namespace.NamespaceContext;
33
34 import java.util.Map;
35 import java.util.Iterator;
36
37 /**
38 * The StartElement interface provides access to information about
39 * start elements. A StartElement is reported for each Start Tag
40 * in the document.
41 *
42 * @version 1.0
43 * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
44 * @since 1.6
45 */
46 public interface StartElement extends XMLEvent {
47
48     /**
49      * Get the name of this event
50      * @return the qualified name of this event
51      */
52     public QName getName();
53 }
```

```
54  /**
55   * Returns an Iterator of non-namespace declared attributes declared on
56   * this START_ELEMENT,
57   * returns an empty iterator if there are no attributes. The
58   * iterator must contain only implementations of the javax.xml.stream.Attribute
59   * interface. Attributes are fundamentally unordered and may not be reported
60   * in any order.
61   *
62   * @return a readonly Iterator over Attribute interfaces, or an
63   * empty iterator
64   */
65  public Iterator getAttributes();
66
67  /**
68   * Returns an Iterator of namespaces declared on this element.
69   * This Iterator does not contain previously declared namespaces
70   * unless they appear on the current START_ELEMENT.
71   * Therefore this list may contain redeclared namespaces and duplicate namespace
72   * declarations. Use the getNamespaceContext() method to get the
73   * current context of namespace declarations.
74   *
75   * <p>The iterator must contain only implementations of the
76   * javax.xml.stream.Namespace interface.
77   *
78   * <p>A Namespace isA Attribute. One
79   * can iterate over a list of namespaces as a list of attributes.
80   * However this method returns only the list of namespaces
81   * declared on this START_ELEMENT and does not
82   * include the attributes declared on this START_ELEMENT.
83   *
84   * Returns an empty iterator if there are no namespaces.
85   *
86   * @return a readonly Iterator over Namespace interfaces, or an
87   * empty iterator
88   *
89   */
90  public Iterator getNamespaces();
91
92  /**
93   * Returns the attribute referred to by this name
94   * @param name the QName of the desired name
95   * @return the attribute corresponding to the name value or null
96   */
97  public Attribute getAttributeByName(QName name);
98
99  /**
100   * Gets a read-only namespace context. If no context is
101   * available this method will return an empty namespace context.
102   * The NamespaceContext contains information about all namespaces
103   * in scope for this StartElement.
104   *
105   * @return the current namespace context
106   */
```

```
107 public NamespaceContext getNamespaceContext();
108
109 /**
110  * Gets the value that the prefix is bound to in the
111  * context of this element. Returns null if
112  * the prefix is not bound in this context
113  * @param prefix the prefix to lookup
114  * @return the uri bound to the prefix or null
115  */
116 public String getNamespaceURI(String prefix);
117 }
```

## 19.14 XMLEvent.java

Secuencia 220: javax/xml/stream/events/XMLEvent.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
29 package javax.xml.stream.events;
30
31 import java.io.Writer;
32 import javax.xml.namespace.QName;
33 /**
34  * This is the base event interface for handling markup events.
35  * Events are value objects that are used to communicate the
36  * XML 1.0 InfoSet to the Application. Events may be cached
37  * and referenced after the parse has completed.
38  *
```

```
39  * @version 1.0
40  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
41  * @see javax.xml.stream.XMLStreamReader
42  * @see Characters
43  * @see ProcessingInstruction
44  * @see StartElement
45  * @see EndElement
46  * @see StartDocument
47  * @see EndDocument
48  * @see EntityReference
49  * @see EntityDeclaration
50  * @see NotationDeclaration
51  * @since 1.6
52  */
53  public interface XMLEvent extends javax.xml.stream.XMLStreamConstants {
54
55      /**
56       * Returns an integer code for this event.
57       * @see #START_ELEMENT
58       * @see #END_ELEMENT
59       * @see #CHARACTERS
60       * @see #ATTRIBUTE
61       * @see #NAMESPACE
62       * @see #PROCESSING_INSTRUCTION
63       * @see #COMMENT
64       * @see #START_DOCUMENT
65       * @see #END_DOCUMENT
66       * @see #DTD
67       */
68      public int getEventType();
69
70      /**
71       * Return the location of this event. The Location
72       * returned from this method is non-volatile and
73       * will retain its information.
74       * @see javax.xml.stream.Location
75       */
76      javax.xml.stream.Location getLocation();
77
78      /**
79       * A utility function to check if this event is a StartElement.
80       * @see StartElement
81       */
82      public boolean isStartElement();
83
84      /**
85       * A utility function to check if this event is an Attribute.
86       * @see Attribute
87       */
88      public boolean isAttribute();
89
90      /**
91       * A utility function to check if this event is a Namespace.
```



```
92     * @see Namespace
93     */
94     public boolean isNamespace();
95
96
97     /**
98     * A utility function to check if this event is a EndElement.
99     * @see EndElement
100    */
101    public boolean isEndElement();
102
103    /**
104    * A utility function to check if this event is an EntityReference.
105    * @see EntityReference
106    */
107    public boolean isEntityReference();
108
109    /**
110    * A utility function to check if this event is a ProcessingInstruction.
111    * @see ProcessingInstruction
112    */
113    public boolean isProcessingInstruction();
114
115    /**
116    * A utility function to check if this event is Characters.
117    * @see Characters
118    */
119    public boolean isCharacters();
120
121    /**
122    * A utility function to check if this event is a StartDocument.
123    * @see StartDocument
124    */
125    public boolean isStartDocument();
126
127    /**
128    * A utility function to check if this event is an EndDocument.
129    * @see EndDocument
130    */
131    public boolean isEndDocument();
132
133    /**
134    * Returns this event as a start element event, may result in
135    * a class cast exception if this event is not a start element.
136    */
137    public StartElement asStartElement();
138
139    /**
140    * Returns this event as an end element event, may result in
141    * a class cast exception if this event is not a end element.
142    */
143    public EndElement asEndElement();
144
```

```

145  /**
146   * Returns this event as Characters, may result in
147   * a class cast exception if this event is not Characters.
148   */
149  public Characters asCharacters();
150
151  /**
152   * This method is provided for implementations to provide
153   * optional type information about the associated event.
154   * It is optional and will return null if no information
155   * is available.
156   */
157  public QName getSchemaType();
158
159  /**
160   * This method will write the XMLEvent as per the XML 1.0 specification as Unicode characters.
161   * No indentation or whitespace should be outputted.
162   *
163   * Any user defined event type SHALL have this method
164   * called when being written to on an output stream.
165   * Built in Event types MUST implement this method,
166   * but implementations MAY choose not call these methods
167   * for optimizations reasons when writing out built in
168   * Events to an output stream.
169   * The output generated MUST be equivalent in terms of the
170   * info set expressed.
171   *
172   * @param writer The writer that will output the data
173   * @throws XMLStreamException if there is a fatal error writing the event
174   */
175  public void writeAsEncodedUnicode(Writer writer)
176      throws javax.xml.stream.XMLStreamException;
177
178  }

```

## 20 Xml Stream Util (javax.xml.stream.util package)

### 20.1 EventReaderDelegate.java

Secuencia 221: javax.xml.stream.util.EventReaderDelegate.java

```

1  /**
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *

```

```
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29  package javax.xml.stream.util;
30
31  import javax.xml.namespace.QName;
32  import javax.xml.namespace.NamespaceContext;
33  import javax.xml.stream.XMLEventReader;
34  import javax.xml.stream.events.XMLEvent;
35  import javax.xml.stream.Location;
36  import javax.xml.stream.XMLStreamException;
37
38  /**
39   * This is the base class for deriving an XMLEventReader
40   * filter.
41   *
42   * This class is designed to sit between an XMLEventReader and an
43   * application's XMLEventReader. By default each method
44   * does nothing but call the corresponding method on the
45   * parent interface.
46   *
47   * @version 1.0
48   * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
49   * @see javax.xml.stream.XMLEventReader
50   * @see StreamReaderDelegate
51   * @since 1.6
52   */
53
54  public class EventReaderDelegate implements XMLEventReader {
55      private XMLEventReader reader;
56
57      /**
58       * Construct an empty filter with no parent.
59       */
60      public EventReaderDelegate(){}
61
62      /**
63       * Construct an filter with the specified parent.
64       * @param reader the parent
65       */
66      public EventReaderDelegate(XMLEventReader reader) {
```

```
67     this.reader = reader;
68 }
69
70 /**
71  * Set the parent of this instance.
72  * @param reader the new parent
73  */
74 public void setParent(XMLEventReader reader) {
75     this.reader = reader;
76 }
77
78 /**
79  * Get the parent of this instance.
80  * @return the parent or null if none is set
81  */
82 public XMLEventReader getParent() {
83     return reader;
84 }
85
86 public XMLEvent nextEvent()
87     throws XMLStreamException
88 {
89     return reader.nextEvent();
90 }
91
92 public Object next() {
93     return reader.next();
94 }
95
96 public boolean hasNext()
97 {
98     return reader.hasNext();
99 }
100
101 public XMLEvent peek()
102     throws XMLStreamException
103 {
104     return reader.peek();
105 }
106
107 public void close()
108     throws XMLStreamException
109 {
110     reader.close();
111 }
112
113 public String getElementText()
114     throws XMLStreamException
115 {
116     return reader.getElementText();
117 }
118
119 public XMLEvent nextTag()
```

```
120     throws XMLStreamException
121     {
122         return reader.nextTag();
123     }
124
125     public Object getProperty(java.lang.String name)
126         throws java.lang.IllegalArgumentException
127     {
128         return reader.getProperty(name);
129     }
130
131     public void remove() {
132         reader.remove();
133     }
134 }
```

## 20.2 StreamReaderDelegate.java

Secuencia 222: javax/xml/stream/util/StreamReaderDelegate.java

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26   * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27   */
28
29  package javax.xml.stream.util;
30
31  import java.io.Reader;
32  import javax.xml.namespace.QName;
33  import javax.xml.namespace.NamespaceContext;
34  import javax.xml.stream.XMLStreamReader;
```

```
35 import javax.xml.stream.Location;
36 import javax.xml.stream.XMLStreamException;
37
38 /**
39  * This is the base class for deriving an XMLStreamReader filter
40  *
41  * This class is designed to sit between an XMLStreamReader and an
42  * application's XMLStreamReader. By default each method
43  * does nothing but call the corresponding method on the
44  * parent interface.
45  *
46  * @version 1.0
47  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
48  * @see javax.xml.stream.XMLStreamReader
49  * @see EventReaderDelegate
50  * @since 1.6
51  */
52
53 public class StreamReaderDelegate implements XMLStreamReader {
54     private XMLStreamReader reader;
55
56     /**
57      * Construct an empty filter with no parent.
58      */
59     public StreamReaderDelegate(){}
60
61     /**
62      * Construct an filter with the specified parent.
63      * @param reader the parent
64      */
65     public StreamReaderDelegate(XMLStreamReader reader) {
66         this.reader = reader;
67     }
68
69     /**
70      * Set the parent of this instance.
71      * @param reader the new parent
72      */
73     public void setParent(XMLStreamReader reader) {
74         this.reader = reader;
75     }
76
77     /**
78      * Get the parent of this instance.
79      * @return the parent or null if none is set
80      */
81     public XMLStreamReader getParent() {
82         return reader;
83     }
84
85     public int next()
86         throws XMLStreamException
87     {
```

```
88     return reader.next();
89 }
90
91 public int nextTag()
92     throws XMLStreamException
93 {
94     return reader.nextTag();
95 }
96
97 public String getElementText()
98     throws XMLStreamException
99 {
100     return reader.getElementText();
101 }
102
103 public void require(int type, String namespaceURI, String localName)
104     throws XMLStreamException
105 {
106     reader.require(type, namespaceURI, localName);
107 }
108
109 public boolean hasNext()
110     throws XMLStreamException
111 {
112     return reader.hasNext();
113 }
114
115 public void close()
116     throws XMLStreamException
117 {
118     reader.close();
119 }
120
121 public String getNamespaceURI(String prefix)
122 {
123     return reader.getNamespaceURI(prefix);
124 }
125
126 public NamespaceContext getNamespaceContext() {
127     return reader.getNamespaceContext();
128 }
129
130 public boolean isStartElement() {
131     return reader.isStartElement();
132 }
133
134 public boolean isEndElement() {
135     return reader.isEndElement();
136 }
137
138 public boolean isCharacters() {
139     return reader.isCharacters();
140 }
```

```
141
142 public boolean isWhiteSpace() {
143     return reader.isWhiteSpace();
144 }
145
146 public String getAttributeValue(String namespaceUri,
147                                 String localName)
148 {
149     return reader.getAttributeValue(namespaceUri,localName);
150 }
151
152 public int getAttributeCount() {
153     return reader.getAttributeCount();
154 }
155
156 public QName getAttributeName(int index) {
157     return reader.getAttributeName(index);
158 }
159
160 public String getAttributePrefix(int index) {
161     return reader.getAttributePrefix(index);
162 }
163
164 public String getAttributeNamespace(int index) {
165     return reader.getAttributeNamespace(index);
166 }
167
168 public String getAttributeLocalName(int index) {
169     return reader.getAttributeLocalName(index);
170 }
171
172 public String getAttributeType(int index) {
173     return reader.getAttributeType(index);
174 }
175
176 public String getAttributeValue(int index) {
177     return reader.getAttributeValue(index);
178 }
179
180 public boolean isAttributeSpecified(int index) {
181     return reader.isAttributeSpecified(index);
182 }
183
184 public int getNamespaceCount() {
185     return reader.getNamespaceCount();
186 }
187
188 public String getNamespacePrefix(int index) {
189     return reader.getNamespacePrefix(index);
190 }
191
192 public String getNamespaceURI(int index) {
193     return reader.getNamespaceURI(index);
```



```
194 }
195
196 public int getEventType() {
197     return reader.getEventType();
198 }
199
200 public String getText() {
201     return reader.getText();
202 }
203
204 public int getTextCharacters(int sourceStart,
205                             char[] target,
206                             int targetStart,
207                             int length)
208     throws XMLStreamException {
209     return reader.getTextCharacters(sourceStart,
210                                    target,
211                                    targetStart,
212                                    length);
213 }
214
215
216 public char[] getTextCharacters() {
217     return reader.getTextCharacters();
218 }
219
220 public int getTextStart() {
221     return reader.getTextStart();
222 }
223
224 public int getTextLength() {
225     return reader.getTextLength();
226 }
227
228 public String getEncoding() {
229     return reader.getEncoding();
230 }
231
232 public boolean hasText() {
233     return reader.hasText();
234 }
235
236 public Location getLocation() {
237     return reader.getLocation();
238 }
239
240 public QName getName() {
241     return reader.getName();
242 }
243
244 public String getLocalName() {
245     return reader.getLocalName();
246 }
```

```
247
248 public boolean hasName() {
249     return reader.hasName();
250 }
251
252 public String getNamespaceURI() {
253     return reader.getNamespaceURI();
254 }
255
256 public String getPrefix() {
257     return reader.getPrefix();
258 }
259
260 public String getVersion() {
261     return reader.getVersion();
262 }
263
264 public boolean isStandalone() {
265     return reader.isStandalone();
266 }
267
268 public boolean standaloneSet() {
269     return reader.standaloneSet();
270 }
271
272 public String getCharacterEncodingScheme() {
273     return reader.getCharacterEncodingScheme();
274 }
275
276 public String getPITarget() {
277     return reader.getPITarget();
278 }
279
280 public String getPIData() {
281     return reader.getPIData();
282 }
283
284 public Object getProperty(String name) {
285     return reader.getProperty(name);
286 }
287 }
```

## 20.3 XMLEventAllocator.java

Secuencia 223: javax/xml/stream/util/XMLEventAllocator.java

```
1 /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26  * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27  */
28
29 package javax.xml.stream.util;
30
31 import javax.xml.stream.events.XMLEvent;
32 import javax.xml.stream.XMLStreamReader;
33 import javax.xml.stream.XMLStreamException;
34
35 /**
36  * This interface defines a class that allows a user to register
37  * a way to allocate events given an XMLStreamReader. An implementation
38  * is not required to use the XMLEventFactory implementation but this
39  * is recommended. The XMLEventAllocator can be set on an XMLInputFactory
40  * using the property "javax.xml.stream allocator"
41  *
42  * @version 1.0
43  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
44  * @see javax.xml.stream.XMLInputFactory
45  * @see javax.xml.stream.XMLEventFactory
46  * @since 1.6
47  */
48 public interface XMLEventAllocator {
49
50     /**
51      * This method creates an instance of the XMLEventAllocator. This
52      * allows the XMLInputFactory to allocate a new instance per reader.
53      */
54     public XMLEventAllocator newInstance();
55
56     /**
57      * This method allocates an event given the current
58      * state of the XMLStreamReader. If this XMLEventAllocator
59      * does not have a one-to-one mapping between reader states
60      * and events this method will return null. This method
61      * must not modify the state of the XMLStreamReader.
```

```
62  * @param reader The XMLStreamReader to allocate from
63  * @return the event corresponding to the current reader state
64  */
65  public XMLEvent allocate(XMLStreamReader reader)
66      throws XMLStreamException;
67
68  /**
69   * This method allocates an event or set of events
70   * given the current
71   * state of the XMLStreamReader and adds the event
72   * or set of events to the
73   * consumer that was passed in. This method can be used
74   * to expand or contract reader states into event states.
75   * This method may modify the state of the XMLStreamReader.
76   * @param reader The XMLStreamReader to allocate from
77   * @param consumer The XMLEventConsumer to add to.
78   */
79  public void allocate(XMLStreamReader reader, XMLEventConsumer consumer)
80      throws XMLStreamException;
81
82 }
```

## 20.4 XMLEventConsumer.java

Secuencia 224: javax/xml/stream/util/XMLEventConsumer.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 * Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
27 */
28
```

```

29 package javax.xml.stream.util;
30
31 import javax.xml.stream.events.XMLEvent;
32 import javax.xml.stream.XMLStreamException;
33
34 /**
35  * This interface defines an event consumer interface. The contract of the
36  * of a consumer is to accept the event. This interface can be used to
37  * mark an object as able to receive events. Add may be called several
38  * times in immediate succession so a consumer must be able to cache
39  * events it hasn't processed yet.
40  *
41  * @version 1.0
42  * @author Copyright (c) 2009 by Oracle Corporation. All Rights Reserved.
43  * @since 1.6
44  */
45 public interface XMLEventConsumer {
46
47     /**
48      * This method adds an event to the consumer. Calling this method
49      * invalidates the event parameter. The client application should
50      * discard all references to this event upon calling add.
51      * The behavior of an application that continues to use such references
52      * is undefined.
53      *
54      * @param event the event to add, may not be null
55      */
56     public void add(XMLEvent event)
57         throws XMLStreamException;
58 }

```

## 21 Xml Transform (javax.xml.transform package)

### 21.1 ErrorListener.java

Secuencia 225: javax/xml/transform/ErrorListener.java

```

1  /*
2   * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *

```

```

18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.transform;
27
28 /**
29  * <p>To provide customized error handling, implement this interface and
30  * use the setErrorListener method to register an instance of the
31  * implementation with the {@link javax.xml.transform.Transformer}. The
32  * Transformer then reports all errors and warnings through this
33  * interface.</p>
34  *
35  * <p>If an application does not register its own custom
36  * ErrorListener, the default ErrorListener
37  * is used which reports all warnings and errors to System.err
38  * and does not throw any Exceptions.
39  * Applications are strongly encouraged to register and use
40  * ErrorListeners that insure proper behavior for warnings and
41  * errors.</p>
42  *
43  * <p>For transformation errors, a Transformer must use this
44  * interface instead of throwing an Exception: it is up to the
45  * application to decide whether to throw an Exception for
46  * different types of errors and warnings. Note however that the
47  * Transformer is not required to continue with the transformation
48  * after a call to {@link #fatalError(TransformerException exception)}.</p>
49  *
50  * <p>Transformers may use this mechanism to report XML parsing
51  * errors as well as transformation errors.</p>
52  */
53 public interface ErrorListener {
54
55     /**
56      * Receive notification of a warning.
57      *
58      * <p>{@link javax.xml.transform.Transformer} can use this method to report
59      * conditions that are not errors or fatal errors. The default behaviour
60      * is to take no action.</p>
61      *
62      * <p>After invoking this method, the Transformer must continue with
63      * the transformation. It should still be possible for the
64      * application to process the document through to the end.</p>
65      *
66      * @param exception The warning information encapsulated in a
67      *                  transformer exception.
68      *
69      * @throws javax.xml.transform.TransformerException if the application
70      * chooses to discontinue the transformation.

```

```

71      *
72      * @see javax.xml.transform.TransformerException
73      */
74      public abstract void warning(TransformerException exception)
75          throws TransformerException;
76
77      /**
78       * Receive notification of a recoverable error.
79       *
80       * <p>The transformer must continue to try and provide normal transformation
81       * after invoking this method. It should still be possible for the
82       * application to process the document through to the end if no other errors
83       * are encountered.</p>
84       *
85       * @param exception The error information encapsulated in a
86       *                  transformer exception.
87       *
88       * @throws javax.xml.transform.TransformerException if the application
89       * chooses to discontinue the transformation.
90       *
91       * @see javax.xml.transform.TransformerException
92       */
93      public abstract void error(TransformerException exception)
94          throws TransformerException;
95
96      /**
97       * <p>Receive notification of a non-recoverable error.</p>
98       *
99       * <p>The processor may choose to continue, but will not normally
100      * proceed to a successful completion.</p>
101      *
102      * <p>The method should throw an exception if it is unable to
103      * process the error, or if it wishes execution to terminate
104      * immediately. The processor will not necessarily honor this
105      * request.</p>
106      *
107      * @param exception The error information encapsulated in a
108      *                  <code>TransformerException</code>.
109      *
110      * @throws javax.xml.transform.TransformerException if the application
111      * chooses to discontinue the transformation.
112      *
113      * @see javax.xml.transform.TransformerException
114      */
115      public abstract void fatalError(TransformerException exception)
116          throws TransformerException;
117  }

```

## 21.2 FactoryFinder.java

Secuencia 226: javax/xml/transform/FactoryFinder.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.

```

```
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 import java.io.File;
29 import java.lang.reflect.Method;
30 import java.lang.reflect.Modifier;
31 import java.security.AccessController;
32 import java.security.PrivilegedAction;
33 import java.util.Iterator;
34 import java.util.Properties;
35 import java.util.ServiceConfigurationError;
36 import java.util.ServiceLoader;
37
38 /**
39  * <p>Implements pluggable Datatypes.</p>
40  *
41  * <p>This class is duplicated for each JAXP subpackage so keep it in
42  * sync. It is package private for secure class loading.</p>
43  *
44  * @author Santiago.PericasGeertsens@sun.com
45  * @author Huizhe.Wang@oracle.com
46  */
47 class FactoryFinder {
48     private static final String DEFAULT_PACKAGE = "com.sun.org.apache.xalan.internal.";
49
50     /**
51      * Internal debug flag.
52      */
53     private static boolean debug = false;
54
55     /**
```



```
56     * Cache for properties in java.home/lib/jaxp.properties
57     */
58     private final static Properties cacheProps = new Properties();
59
60     /**
61     * Flag indicating if properties from java.home/lib/jaxp.properties
62     * have been cached.
63     */
64     static volatile boolean firstTime = true;
65
66     /**
67     * Security support class use to check access control before
68     * getting certain system resources.
69     */
70     private final static SecuritySupport ss = new SecuritySupport();
71
72     // Define system property "jaxp.debug" to get output
73     static {
74         // Use try/catch block to support applets, which throws
75         // SecurityException out of this code.
76         try {
77             String val = ss.getProperty("jaxp.debug");
78             // Allow simply setting the prop to turn on debug
79             debug = val != null && !"false".equals(val);
80         }
81         catch (SecurityException se) {
82             debug = false;
83         }
84     }
85
86     private static void dPrint(String msg) {
87         if (debug) {
88             System.err.println("JAXP: " + msg);
89         }
90     }
91
92     /**
93     * Attempt to load a class using the class loader supplied. If that fails
94     * and fall back is enabled, the current (i.e. bootstrap) class loader is
95     * tried.
96     *
97     * If the class loader supplied is <code>null</code>, first try using the
98     * context class loader followed by the current (i.e. bootstrap) class
99     * loader.
100    *
101    * Use bootstrap classLoader if cl = null and useBSClsLoader is true
102    */
103    static private Class<?> getProviderClass(String className, ClassLoader cl,
104        boolean doFallback, boolean useBSClsLoader) throws ClassNotFoundException
105    {
106        try {
107            if (cl == null) {
108                if (useBSClsLoader) {
```

```

109         return Class.forName(className, false, FactoryFinder.class.getClassLoader());
110     } else {
111         cl = ss.getContextClassLoader();
112         if (cl == null) {
113             throw new ClassNotFoundException();
114         }
115         else {
116             return Class.forName(className, false, cl);
117         }
118     }
119 }
120 else {
121     return Class.forName(className, false, cl);
122 }
123 }
124 catch (ClassNotFoundException e1) {
125     if (doFallback) {
126         // Use current class loader - should always be bootstrap CL
127         return Class.forName(className, false, FactoryFinder.class.getClassLoader());
128     }
129     else {
130         throw e1;
131     }
132 }
133 }
134
135 /**
136  * Create an instance of a class. Delegates to method
137  * <code>getProviderClass()</code> in order to load the class.
138  *
139  * @param type Base class / Service interface of the factory to
140  *             instantiate.
141  *
142  * @param className Name of the concrete class corresponding to the
143  * service provider
144  *
145  * @param cl <code>ClassLoader</code> used to load the factory class. If <code>null</code>
146  * current <code>Thread</code>'s context classLoader is used to load the factory class.
147  *
148  * @param doFallback True if the current ClassLoader should be tried as
149  * a fallback if the class is not found using cl
150  *
151  * @param useServicesMechanism True use services mechanism
152  */
153 static <T> T newInstance(Class<T> type, String className, ClassLoader cl,
154                         boolean doFallback, boolean useServicesMechanism)
155     throws TransformerFactoryConfigurationError
156 {
157     assert type != null;
158
159     boolean useBSClsLoader = false;
160     // make sure we have access to restricted packages
161     if (System.getSecurityManager() != null) {

```

```

162         if (className != null && className.startsWith(DEFAULT_PACKAGE)) {
163             cl = null;
164             useBSCLsLoader = true;
165         }
166     }
167
168     try {
169         Class<?> providerClass = getProviderClass(className, cl, doFallback, useBSCLsLoader);
170         if (!type.isAssignableFrom(providerClass)) {
171             throw new ClassCastException(className + " cannot be cast to " + type.getName());
172         }
173         Object instance = null;
174         if (!useServicesMechanism) {
175             instance = newInstanceNoServiceLoader(type, providerClass);
176         }
177         if (instance == null) {
178             instance = providerClass.newInstance();
179         }
180         if (debug) { // Extra check to avoid computing cl strings
181             dPrint("created new instance of " + providerClass +
182                 " using ClassLoader: " + cl);
183         }
184         return type.cast(instance);
185     }
186     catch (ClassNotFoundException x) {
187         throw new TransformerFactoryConfigurationError(x,
188             "Provider " + className + " not found");
189     }
190     catch (Exception x) {
191         throw new TransformerFactoryConfigurationError(x,
192             "Provider " + className + " could not be instantiated: " + x);
193     }
194 }
195
196 /**
197  * Try to construct using newTransformerFactoryNoServiceLoader
198  * method if available.
199  */
200 private static <T> T newInstanceNoServiceLoader(Class<T> type, Class<?> providerClass) {
201     // Retain maximum compatibility if no security manager.
202     if (System.getSecurityManager() == null) {
203         return null;
204     }
205     try {
206         final Method creationMethod =
207             providerClass.getDeclaredMethod(
208                 "newTransformerFactoryNoServiceLoader"
209             );
210         final int modifiers = creationMethod.getModifiers();
211
212         // Do not call the method if it's not public static.
213         if (!Modifier.isPublic(modifiers) || !Modifier.isStatic(modifiers)) {
214             return null;

```

```

215     }
216
217     // Only call the method if it's declared to return an instance of
218     // TransformerFactory
219     final Class<?> returnType = creationMethod.getReturnType();
220     if (type.isAssignableFrom(returnType)) {
221         final Object result = creationMethod.invoke(null, (Object[])null);
222         return type.cast(result);
223     } else {
224         // This should not happen, as
225         // TransformerFactoryImpl.newTransformerFactoryNoServiceLoader is
226         // declared to return TransformerFactory.
227         throw new ClassCastException(returnType + " cannot be cast to " + type);
228     }
229 } catch (ClassCastException e) {
230     throw new TransformerFactoryConfigurationError(e, e.getMessage());
231 } catch (NoSuchMethodException exc) {
232     return null;
233 } catch (Exception exc) {
234     return null;
235 }
236 }
237
238 /**
239  * Finds the implementation Class object in the specified order. Main
240  * entry point.
241  * @return Class object of factory, never null
242  *
243  * @param type                Base class / Service interface of the
244  *                             factory to find.
245  *
246  * @param fallbackClassName    Implementation class name, if nothing else
247  *                             is found. Use null to mean no fallback.
248  *
249  * Package private so this code can be shared.
250  */
251 static <T> T find(Class<T> type, String fallbackClassName)
252     throws TransformerFactoryConfigurationError
253 {
254     assert type != null;
255
256     final String factoryId = type.getName();
257
258     dPrint("find factoryId =" + factoryId);
259     // Use the system property first
260     try {
261         String systemProp = ss.getProperty(factoryId);
262         if (systemProp != null) {
263             dPrint("found system property, value=" + systemProp);
264             return newInstance(type, systemProp, null, true, true);
265         }
266     }
267     catch (SecurityException se) {

```

```

268         if (debug) se.printStackTrace();
269     }
270
271     // try to read from $java.home/lib/jaxp.properties
272     try {
273         if (firstTime) {
274             synchronized (cacheProps) {
275                 if (firstTime) {
276                     String configFile = ss.getProperty("java.home") + File.separator +
277                         "lib" + File.separator + "jaxp.properties";
278                     File f = new File(configFile);
279                     firstTime = false;
280                     if (ss.exists(f)) {
281                         dPrint("Read properties file "+f);
282                         cacheProps.load(ss.getInputStream(f));
283                     }
284                 }
285             }
286         }
287         final String factoryClassName = cacheProps.getProperty(factoryId);
288
289         if (factoryClassName != null) {
290             dPrint("found in $java.home/jaxp.properties, value=" + factoryClassName);
291             return newInstance(type, factoryClassName, null, true, true);
292         }
293     }
294     catch (Exception ex) {
295         if (debug) ex.printStackTrace();
296     }
297
298     // Try Jar Service Provider Mechanism
299     T provider = findServiceProvider(type);
300     if (provider != null) {
301         return provider;
302     }
303     if (fallbackClassName == null) {
304         throw new TransformerFactoryConfigurationError(null,
305             "Provider for " + factoryId + " cannot be found");
306     }
307
308     dPrint("loaded from fallback value: " + fallbackClassName);
309     return newInstance(type, fallbackClassName, null, true, true);
310 }
311
312 /*
313  * Try to find provider using the ServiceLoader.
314  *
315  * @param type Base class / Service interface of the factory to find.
316  *
317  * @return instance of provider class if found or null
318  */
319 private static <T> T findServiceProvider(final Class<T> type)
320     throws TransformerFactoryConfigurationError

```

```

321 {
322     try {
323         return AccessController.doPrivileged(new PrivilegedAction<T>() {
324             public T run() {
325                 final ServiceLoader<T> serviceLoader = ServiceLoader.load(type);
326                 final Iterator<T> iterator = serviceLoader.iterator();
327                 if (iterator.hasNext()) {
328                     return iterator.next();
329                 } else {
330                     return null;
331                 }
332             }
333         });
334     } catch (ServiceConfigurationError e) {
335         // It is not possible to wrap an error directly in
336         // FactoryConfigurationError - so we need to wrap the
337         // ServiceConfigurationError in a RuntimeException.
338         // The alternative would be to modify the logic in
339         // FactoryConfigurationError to allow setting a
340         // Throwable as the cause, but that could cause
341         // compatibility issues down the road.
342         final RuntimeException x = new RuntimeException(
343             "Provider for " + type + " cannot be created", e);
344         final TransformerFactoryConfigurationError error =
345             new TransformerFactoryConfigurationError(x, x.getMessage());
346         throw error;
347     }
348 }
349 }

```

## 21.3 OutputKeys.java

Secuencia 227: javax/xml/transform/OutputKeys.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```

```

21  *
22  *
23  *
24  */
25
26  package javax.xml.transform;
27
28  /**
29   * Provides string constants that can be used to set
30   * output properties for a Transformer, or to retrieve
31   * output properties from a Transformer or Templates object.
32   * <p>All the fields in this class are read-only.</p>
33   *
34   * @see <a href="http://www.w3.org/TR/xslt#output">
35   * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
36   */
37  public class OutputKeys {
38
39      /**
40       * Default constructor is private on purpose. This class is
41       * only for static variable access, and should never be constructed.
42       */
43      private OutputKeys() { }
44
45      /**
46       * method = "xml" | "html" | "text" | <var>expanded name</var>.
47       *
48       * <p>The value of the method property identifies the overall method that
49       * should be used for outputting the result tree. Other non-namespaced
50       * values may be used, such as "xhtml", but, if accepted, the handling
51       * of such values is implementation defined. If any of the method values
52       * are not accepted and are not namespace qualified,
53       * then {@link javax.xml.transform.Transformer#setOutputProperty}
54       * or {@link javax.xml.transform.Transformer#setOutputProperties} will
55       * throw a {@link java.lang.IllegalArgumentException}.</p>
56       *
57       * @see <a href="http://www.w3.org/TR/xslt#output">
58       * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
59       */
60      public static final String METHOD = "method";
61
62      /**
63       * version = <var>nmtoken</var>.
64       *
65       * <p><code>version</code> specifies the version of the output
66       * method.</p>
67       * <p>When the output method is "xml", the version value specifies the
68       * version of XML to be used for outputting the result tree. The default
69       * value for the xml output method is 1.0. When the output method is
70       * "html", the version value indicates the version of the HTML.
71       * The default value for the xml output method is 4.0, which specifies
72       * that the result should be output as HTML conforming to the HTML 4.0
73       * Recommendation [HTML]. If the output method is "text", the version

```

```

74     * property is ignored.</p>
75     * @see <a href="http://www.w3.org/TR/xslt#output">
76     * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
77     */
78     public static final String VERSION = "version";
79
80     /**
81     * encoding = <var>string</var>.
82     *
83     * <p><code>encoding</code> specifies the preferred character
84     * encoding that the Transformer should use to encode sequences of
85     * characters as sequences of bytes. The value of the encoding property should be
86     * treated case-insensitively. The value must only contain characters in
87     * the range #x21 to #x7E (i.e., printable ASCII characters). The value
88     * should either be a <code>charset</code> registered with the Internet
89     * Assigned Numbers Authority <a href="http://www.iana.org/">[IANA]</a>,
90     * <a href="http://www.ietf.org/rfc/rfc2278.txt">[RFC2278]</a>
91     * or start with <code>X-</code>.</p>
92     * @see <a href="http://www.w3.org/TR/xslt#output">
93     * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
94     */
95     public static final String ENCODING = "encoding";
96
97     /**
98     * omit-xml-declaration = "yes" | "no".
99     *
100    * <p><code>omit-xml-declaration</code> specifies whether the XSLT
101    * processor should output an XML declaration; the value must be
102    * <code>yes</code> or <code>no</code>.</p>
103    * @see <a href="http://www.w3.org/TR/xslt#output">
104    * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
105    */
106    public static final String OMIT_XML_DECLARATION = "omit-xml-declaration";
107
108    /**
109    * standalone = "yes" | "no".
110    *
111    * <p><code>standalone</code> specifies whether the Transformer
112    * should output a standalone document declaration; the value must be
113    * <code>yes</code> or <code>no</code>.</p>
114    * @see <a href="http://www.w3.org/TR/xslt#output">
115    * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
116    */
117    public static final String STANDALONE = "standalone";
118
119    /**
120    * doctype-public = <var>string</var>.
121    * <p>See the documentation for the {@link #DOCTYPE_SYSTEM} property
122    * for a description of what the value of the key should be.</p>
123    *
124    * @see <a href="http://www.w3.org/TR/xslt#output">
125    * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
126    */

```



```

127 public static final String DOCTYPE_PUBLIC = "doctype-public";
128
129 /**
130  * doctype-system = <var>string</var>.
131  * <p><code>doctype-system</code> specifies the system identifier
132  * to be used in the document type declaration.</p>
133  * <p>If the doctype-system property is specified, the xml output method
134  * should output a document type declaration immediately before the first
135  * element. The name following &lt;!DOCTYPE should be the name of the first
136  * element. If doctype-public property is also specified, then the xml
137  * output method should output PUBLIC followed by the public identifier
138  * and then the system identifier; otherwise, it should output SYSTEM
139  * followed by the system identifier. The internal subset should be empty.
140  * The value of the doctype-public property should be ignored unless the doctype-system
141  * property is specified.</p>
142  * <p>If the doctype-public or doctype-system properties are specified,
143  * then the html output method should output a document type declaration
144  * immediately before the first element. The name following &lt;!DOCTYPE
145  * should be HTML or html. If the doctype-public property is specified,
146  * then the output method should output PUBLIC followed by the specified
147  * public identifier; if the doctype-system property is also specified,
148  * it should also output the specified system identifier following the
149  * public identifier. If the doctype-system property is specified but
150  * the doctype-public property is not specified, then the output method
151  * should output SYSTEM followed by the specified system identifier.</p>
152  *
153  * <p><code>doctype-system</code> specifies the system identifier
154  * to be used in the document type declaration.</p>
155  * @see <a href="http://www.w3.org/TR/xslt#output">
156  * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
157  */
158 public static final String DOCTYPE_SYSTEM = "doctype-system";
159
160 /**
161  * cdata-section-elements = <var>expanded names</var>.
162  *
163  * <p><code>cdata-section-elements</code> specifies a whitespace delimited
164  * list of the names of elements whose text node children should be output
165  * using CDATA sections. Note that these names must use the format
166  * described in the section Qualified Name Representation in
167  * {@link javax.xml.transform}.</p>
168  *
169  * @see <a href="http://www.w3.org/TR/xslt#output">
170  * section 16 of the XSL Transformations (XSLT) W3C Recommendation.</a>
171  */
172 public static final String CDATA_SECTION_ELEMENTS =
173     "cdata-section-elements";
174
175 /**
176  * indent = "yes" | "no".
177  *
178  * <p><code>indent</code> specifies whether the Transformer may
179  * add additional whitespace when outputting the result tree; the value

```

```

180     * must be <code>yes</code> or <code>no</code>. </p>
181     * @see <a href="http://www.w3.org/TR/xslt#output">
182     * section 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
183     */
184     public static final String INDENT = "indent";
185
186     /**
187     * media-type = <var>string</var>.
188     *
189     * <p><code>media-type</code> specifies the media type (MIME
190     * content type) of the data that results from outputting the result
191     * tree. The <code>charset</code> parameter should not be specified
192     * explicitly; instead, when the top-level media type is
193     * <code>text</code>, a <code>charset</code> parameter should be added
194     * according to the character encoding actually used by the output
195     * method. </p>
196     * @see <a href="http://www.w3.org/TR/xslt#output">s
197     * ection 16 of the XSL Transformations (XSLT) W3C Recommendation</a>
198     */
199     public static final String MEDIA_TYPE = "media-type";
200 }

```

## 21.4 Result.java

Secuencia 228: javax.xml.transform.Result.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 /**

```

```

29  * <p>An object that implements this interface contains the information
30  * needed to build a transformation result tree.</p>
31  *
32  * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>
33  */
34  public interface Result {
35
36      /**
37       * The name of the processing instruction that is sent if the
38       * result tree disables output escaping.
39       *
40       * <p>Normally, result tree serialization escapes & and < (and
41       * possibly other characters) when outputting text nodes.
42       * This ensures that the output is well-formed XML. However,
43       * it is sometimes convenient to be able to produce output that is
44       * almost, but not quite well-formed XML; for example,
45       * the output may include ill-formed sections that will
46       * be transformed into well-formed XML by a subsequent non-XML aware
47       * process. If a processing instruction is sent with this name,
48       * serialization should be output without any escaping. </p>
49       *
50       * <p>Result DOM trees may also have PI_DISABLE_OUTPUT_ESCAPING and
51       * PI_ENABLE_OUTPUT_ESCAPING inserted into the tree.</p>
52       *
53       * @see <a href="http://www.w3.org/TR/xslt#disable-output-escaping">disable-output-escaping in
54       *      XSLT Specification</a>
55       */
56      public static final String PI_DISABLE_OUTPUT_ESCAPING =
57          "javax.xml.transform.disable-output-escaping";
58
59      /**
60       * The name of the processing instruction that is sent
61       * if the result tree enables output escaping at some point after having
62       * received a PI_DISABLE_OUTPUT_ESCAPING processing instruction.
63       *
64       * @see <a href="http://www.w3.org/TR/xslt#disable-output-escaping">disable-output-escaping in
65       *      XSLT Specification</a>
66       */
67      public static final String PI_ENABLE_OUTPUT_ESCAPING =
68          "javax.xml.transform.enable-output-escaping";
69
70      /**
71       * Set the system identifier for this Result.
72       *
73       * <p>If the Result is not to be written to a file, the system identifier is optional.
74       * The application may still want to provide one, however, for use in error messages
75       * and warnings, or to resolve relative output identifiers.</p>
76       *
77       * @param systemId The system identifier as a URI string.
78       */
79      public void setSystemId(String systemId);
80
81      /**

```

```

80     * Get the system identifier that was set with setSystemId.
81     *
82     * @return The system identifier that was set with setSystemId,
83     * or null if setSystemId was not called.
84     */
85     public String getSystemId();
86 }

```

## 21.5 SecuritySupport.java

Secuencia 229: javax/xml/transform/SecuritySupport.java

```

1  /*
2  * Copyright (c) 2004, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 import java.security.*;
29 import java.net.*;
30 import java.io.*;
31 import java.util.*;
32
33 /**
34  * This class is duplicated for each JAXP subpackage so keep it in sync.
35  * It is package private and therefore is not exposed as part of the JAXP
36  * API.
37  *
38  * Security related methods that only work on J2SE 1.2 and newer.
39  */
40 class SecuritySupport {
41
42

```

```
43     ClassLoader getContextClassLoader() throws SecurityException{
44         return (ClassLoader)
45             AccessController.doPrivileged(new PrivilegedAction() {
46                 public Object run() {
47                     ClassLoader cl = null;
48                     //try {
49                         cl = Thread.currentThread().getContextClassLoader();
50                     //} catch (SecurityException ex) { }
51                     if (cl == null)
52                         cl = ClassLoader.getSystemClassLoader();
53                     return cl;
54                 }
55             });
56     }
57
58     String getSystemProperty(final String propName) {
59         return (String)
60             AccessController.doPrivileged(new PrivilegedAction() {
61                 public Object run() {
62                     return System.getProperty(propName);
63                 }
64             });
65     }
66
67     FileInputStream getFileInputStream(final File file)
68         throws FileNotFoundException
69     {
70         try {
71             return (FileInputStream)
72                 AccessController.doPrivileged(new PrivilegedExceptionAction() {
73                     public Object run() throws FileNotFoundException {
74                         return new FileInputStream(file);
75                     }
76                 });
77         } catch (PrivilegedActionException e) {
78             throw (FileNotFoundException)e.getException();
79         }
80     }
81
82     InputStream getResourceAsStream(final ClassLoader cl,
83                                     final String name)
84     {
85         return (InputStream)
86             AccessController.doPrivileged(new PrivilegedAction() {
87                 public Object run() {
88                     InputStream ris;
89                     if (cl == null) {
90                         ris = Object.class.getResourceAsStream(name);
91                     } else {
92                         ris = cl.getResourceAsStream(name);
93                     }
94                     return ris;
95                 }
96             });
97     }
```

```

96         });
97     }
98
99     boolean doesFileExist(final File f) {
100         return ((Boolean)
101             AccessController.doPrivileged(new PrivilegedAction() {
102                 public Object run() {
103                     return new Boolean(f.exists());
104                 }
105             })).booleanValue();
106     }
107
108 }

```

## 21.6 Source.java

Secuencia 230: javax.xml.transform/Source.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 /**
29  * An object that implements this interface contains the information
30  * needed to act as source input (XML source or transformation instructions).
31  */
32 public interface Source {
33
34     /**
35      * Set the system identifier for this Source.
36      *

```

```

37     * <p>The system identifier is optional if the source does not
38     * get its data from a URL, but it may still be useful to provide one.
39     * The application can use a system identifier, for example, to resolve
40     * relative URIs and to include in error messages and warnings.</p>
41     *
42     * @param systemId The system identifier as a URL string.
43     */
44     public void setSystemId(String systemId);
45
46     /**
47     * Get the system identifier that was set with setSystemId.
48     *
49     * @return The system identifier that was set with setSystemId, or null
50     *         if setSystemId was not called.
51     */
52     public String getSystemId();
53 }

```

## 21.7 SourceLocator.java

Secuencia 231: javax/xml/transform/SourceLocator.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 /**
29 * This interface is primarily for the purposes of reporting where
30 * an error occurred in the XML source or transformation instructions.
31 */
32 public interface SourceLocator {

```

```
33
34 /**
35  * Return the public identifier for the current document event.
36  *
37  * <p>The return value is the public identifier of the document
38  * entity or of the external parsed entity in which the markup that
39  * triggered the event appears.</p>
40  *
41  * @return A string containing the public identifier, or
42  *         null if none is available.
43  * @see #getSystemId
44  */
45 public String getPublicId();
46
47 /**
48  * Return the system identifier for the current document event.
49  *
50  * <p>The return value is the system identifier of the document
51  * entity or of the external parsed entity in which the markup that
52  * triggered the event appears.</p>
53  *
54  * <p>If the system identifier is a URL, the parser must resolve it
55  * fully before passing it to the application.</p>
56  *
57  * @return A string containing the system identifier, or null
58  *         if none is available.
59  * @see #getPublicId
60  */
61 public String getSystemId();
62
63 /**
64  * Return the line number where the current document event ends.
65  *
66  * <p><strong>Warning:</strong> The return value from the method
67  * is intended only as an approximation for the sake of error
68  * reporting; it is not intended to provide sufficient information
69  * to edit the character content of the original XML document.</p>
70  *
71  * <p>The return value is an approximation of the line number
72  * in the document entity or external parsed entity where the
73  * markup that triggered the event appears.</p>
74  *
75  * @return The line number, or -1 if none is available.
76  * @see #getColumnNumber
77  */
78 public int getLineNumber();
79
80 /**
81  * Return the character position where the current document event ends.
82  *
83  * <p><strong>Warning:</strong> The return value from the method
84  * is intended only as an approximation for the sake of error
85  * reporting; it is not intended to provide sufficient information
```



```

86      * to edit the character content of the original XML document.</p>
87      *
88      * <p>The return value is an approximation of the column number
89      * in the document entity or external parsed entity where the
90      * markup that triggered the event appears.</p>
91      *
92      * @return The column number, or -1 if none is available.
93      * @see #getLineNumber
94      */
95      public int getColumnNumber();
96  }

```

## 21.8 Templates.java

Secuencia 232: javax/xml/transform/Templates.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 import java.util.Properties;
29
30
31
32
33 /**
34  * An object that implements this interface is the runtime representation of processed
35  * transformation instructions.
36  *
37  * <p>Templates must be threadsafe for a given instance
38  * over multiple threads running concurrently, and may

```

```

39  * be used multiple times in a given session.</p>
40  */
41  public interface Templates {
42
43      /**
44       * Create a new transformation context for this Templates object.
45       *
46       * @return A valid non-null instance of a Transformer.
47       *
48       * @throws TransformerConfigurationException if a Transformer can not be created.
49       */
50      Transformer newTransformer() throws TransformerConfigurationException;
51
52      /**
53       * Get the properties corresponding to the effective xsl:output element.
54       * The object returned will
55       * be a clone of the internal values. Accordingly, it can be mutated
56       * without mutating the Templates object, and then handed in to
57       * {@link javax.xml.transform.Transformer#setOutputProperties}.
58       *
59       * <p>The properties returned should contain properties set by the stylesheet,
60       * and these properties are "defaulted" by default properties specified by
61       * <a href="http://www.w3.org/TR/xslt#output">section 16 of the
62       * XSL Transformations (XSLT) W3C Recommendation</a>. The properties that
63       * were specifically set by the stylesheet should be in the base
64       * Properties list, while the XSLT default properties that were not
65       * specifically set should be in the "default" Properties list. Thus,
66       * getOutputProperties().getProperty(String key) will obtain any
67       * property in that was set by the stylesheet, <em>or</em> the default
68       * properties, while
69       * getOutputProperties().get(String key) will only retrieve properties
70       * that were explicitly set in the stylesheet.</p>
71       *
72       * <p>For XSLT,
73       * <a href="http://www.w3.org/TR/xslt#attribute-value-templates">Attribute
74       * Value Templates</a> attribute values will
75       * be returned unexpanded (since there is no context at this point). The
76       * namespace prefixes inside Attribute Value Templates will be unexpanded,
77       * so that they remain valid XPath values.</p>
78       *
79       * @return A Properties object, never null.
80       */
81      Properties getOutputProperties();
82  }

```

## 21.9 Transformer.java

Secuencia 233: javax/xml/transform/Transformer.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *

```

```
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 import java.util.Properties;
29
30 /**
31  * An instance of this abstract class can transform a
32  * source tree into a result tree.
33  *
34  * <p>An instance of this class can be obtained with the
35  * {@link TransformerFactory#newTransformer TransformerFactory.newTransformer}
36  * method. This instance may then be used to process XML from a
37  * variety of sources and write the transformation output to a
38  * variety of sinks.</p>
39  *
40  * <p>An object of this class may not be used in multiple threads
41  * running concurrently. Different Transformers may be used
42  * concurrently by different threads.</p>
43  *
44  * <p>A Transformer may be used multiple times. Parameters and
45  * output properties are preserved across transformations.</p>
46  *
47  * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>
48  */
49 public abstract class Transformer {
50
51     /**
52      * Default constructor is protected on purpose.
53      */
54     protected Transformer() { }
55
56     /**
57      * <p>Reset this Transformer to its original configuration.</p>
58      *
```

```

59      * <p><code>Transformer</code> is reset to the same state as when it was created with
60      * {@link TransformerFactory#newTransformer()},
61      * {@link TransformerFactory#newTransformer(Source source)} or
62      * {@link Templates#newTransformer()}.
63      * <code>reset()</code> is designed to allow the reuse of existing <code>Transformer</code>
64      * s
65      * thus saving resources associated with the creation of new <code>Transformer</code>s.</p>
66      * <p>The reset <code>Transformer</code> is not guaranteed to have the same {@link
67      * URIResolver}
68      * or {@link ErrorListener} <code>Object</code>s, e.g. {@link Object#equals(Object obj)}.
69      * It is guaranteed to have a functionally equal <code>URIResolver</code>
70      * and <code>ErrorListener</code>.</p>
71      *
72      * @throws UnsupportedOperationException When implementation does not
73      * override this method.
74      *
75      * @since 1.5
76      */
77      public void reset() {
78          // implementors should override this method
79          throw new UnsupportedOperationException(
80              "This Transformer, \"" + this.getClass().getName() + "\", does not support
81              the reset functionality."
82              + " Specification \"" + this.getClass().getPackage().getSpecificationTitle
83              () + "\"
84              + " version \"" + this.getClass().getPackage().getSpecificationVersion() +
85              "\"
86              );
87      }
88
89      /**
90      * <p>Transform the XML <code>Source</code> to a <code>Result</code>.
91      * Specific transformation behavior is determined by the settings of the
92      * <code>TransformerFactory</code> in effect when the
93      * <code>Transformer</code> was instantiated and any modifications made to
94      * the <code>Transformer</code> instance.</p>
95      *
96      * <p>An empty <code>Source</code> is represented as an empty document
97      * as constructed by {@link javax.xml.parsers.DocumentBuilder#newDocument()}.
98      * The result of transforming an empty <code>Source</code> depends on
99      * the transformation behavior; it is not always an empty
100      * <code>Result</code>.</p>
101      *
102      * @param xmlSource The XML input to transform.
103      * @param outputTarget The <code>Result</code> of transforming the
104      * <code>xmlSource</code>.
105      *
106      * @throws TransformerException If an unrecoverable error occurs
107      * during the course of the transformation.
108      */
109      public abstract void transform(Source xmlSource, Result outputTarget)

```

```

107         throws TransformerException;
108
109     /**
110     * Add a parameter for the transformation.
111     *
112     * <p>Pass a qualified name as a two-part string, the namespace URI
113     * enclosed in curly braces ({}), followed by the local name. If the
114     * name has a null URL, the String only contain the local name. An
115     * application can safely check for a non-null URI by testing to see if the
116     * first character of the name is a '{' character.</p>
117     * <p>For example, if a URI and local name were obtained from an element
118     * defined with <xyz:foo
119     * xmlns:xyz="http://xyz.foo.com/yada/baz.html">,
120     * then the qualified name would be "{http://xyz.foo.com/yada/baz.html}foo".
121     * Note that no prefix is used.</p>
122     *
123     * @param name The name of the parameter, which may begin with a
124     * namespace URI in curly braces ({}).
125     * @param value The value object. This can be any valid Java object. It is
126     * up to the processor to provide the proper object coercion or to simply
127     * pass the object on for use in an extension.
128     *
129     * @throws NullPointerException If value is null.
130     */
131     public abstract void setParameter(String name, Object value);
132
133     /**
134     * Get a parameter that was explicitly set with setParameter.
135     *
136     * <p>This method does not return a default parameter value, which
137     * cannot be determined until the node context is evaluated during
138     * the transformation process.
139     *
140     * @param name of <code>Object</code> to get
141     *
142     * @return A parameter that has been set with setParameter.
143     */
144     public abstract Object getParameter(String name);
145
146     /**
147     * <p>Set a list of parameters.</p>
148     *
149     * <p>Note that the list of parameters is specified as a
150     * <code>Properties</code> <code>Object</code> which limits the parameter
151     * values to <code>String</code>s. Multiple calls to
152     * {@link #setParameter(String name, Object value)} should be used when the
153     * desired values are non-<code>String</code> <code>Object</code>s.
154     * The parameter names should conform as specified in
155     * {@link #setParameter(String name, Object value)}.
156     * An <code>IllegalArgumentException</code> is thrown if any names do not
157     * conform.</p>
158     *
159     * <p>New parameters in the list are added to any existing parameters.

```

```

160     * If the name of a new parameter is equal to the name of an existing
161     * parameter as determined by {@link java.lang.Object#equals(Object obj)},
162     * the existing parameter is set to the new value.</p>
163     *
164     * @param params Parameters to set.
165     *
166     * @throws IllegalArgumentException If any parameter names do not conform
167     * to the naming rules.
168     */
169
170 /**
171  * Clear all parameters set with setParameter.
172  */
173 public abstract void clearParameters();
174
175 /**
176  * Set an object that will be used to resolve URIs used in
177  * document().
178  *
179  * <p>If the resolver argument is null, the URIResolver value will
180  * be cleared and the transformer will no longer have a resolver.</p>
181  *
182  * @param resolver An object that implements the URIResolver interface,
183  * or null.
184  */
185 public abstract void setURIResolver(URIResolver resolver);
186
187 /**
188  * Get an object that will be used to resolve URIs used in
189  * document().
190  *
191  * @return An object that implements the URIResolver interface,
192  * or null.
193  */
194 public abstract URIResolver getURIResolver();
195
196 /**
197  * Set the output properties for the transformation. These
198  * properties will override properties set in the Templates
199  * with xsl:output.
200  *
201  * <p>If argument to this function is null, any properties
202  * previously set are removed, and the value will revert to the value
203  * defined in the templates object.</p>
204  *
205  * <p>Pass a qualified property key name as a two-part string, the namespace
206  * URI enclosed in curly braces ({}), followed by the local name. If the
207  * name has a null URL, the String only contain the local name. An
208  * application can safely check for a non-null URI by testing to see if the
209  * first character of the name is a '{' character.</p>
210  * <p>For example, if a URI and local name were obtained from an element
211  * defined with <xyz:foo
212  * xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>,

```

```

213 * then the qualified name would be "{http://xyz.foo.com/yada/baz.html}foo".
214 * Note that no prefix is used.</p>
215 * An <code>IllegalArgumentException</code> is thrown if any of the
216 * argument keys are not recognized and are not namespace qualified.
217 *
218 * @param oformat A set of output properties that will be
219 * used to override any of the same properties in affect
220 * for the transformation.
221 *
222 * @throws IllegalArgumentException When keys are not recognized and
223 * are not namespace qualified.
224 *
225 * @see javax.xml.transform.OutputKeys
226 * @see java.util.Properties
227 *
228 */
229 public abstract void setOutputProperties(Properties oformat);
230
231 /**
232 * <p>Get a copy of the output properties for the transformation.</p>
233 *
234 * <p>The properties returned should contain properties set by the user,
235 * and properties set by the stylesheet, and these properties
236 * are "defaulted" by default properties specified by
237 * <a href="http://www.w3.org/TR/xslt#output">section 16 of the
238 * XSL Transformations (XSLT) W3C Recommendation</a>. The properties that
239 * were specifically set by the user or the stylesheet should be in the base
240 * Properties list, while the XSLT default properties that were not
241 * specifically set should be the default Properties list. Thus,
242 * <code>getOutputProperties().getProperty(String key)</code> will obtain any
243 * property in that was set by {@link #setOutputProperty},
244 * {@link #setOutputProperties}, in the stylesheet, <em>or</em> the default
245 * properties, while
246 * <code>getOutputProperties().get(String key)</code> will only retrieve properties
247 * that were explicitly set by {@link #setOutputProperty},
248 * {@link #setOutputProperties}, or in the stylesheet.</p>
249 *
250 * <p>Note that mutation of the Properties object returned will not
251 * effect the properties that the transformer contains.</p>
252 *
253 * <p>If any of the argument keys are not recognized and are not
254 * namespace qualified, the property will be ignored and not returned.
255 * In other words the behaviour is not orthogonal with
256 * {@link #setOutputProperties setOutputProperties}.</p>
257 *
258 * @return A copy of the set of output properties in effect for
259 * the next transformation.
260 *
261 * @see javax.xml.transform.OutputKeys
262 * @see java.util.Properties
263 * @see <a href="http://www.w3.org/TR/xslt#output">
264 * XSL Transformations (XSLT) Version 1.0</a>
265 */

```

```

266 public abstract Properties getOutputProperties();
267
268 /**
269  * Set an output property that will be in effect for the
270  * transformation.
271  *
272  * <p>Pass a qualified property name as a two-part string, the namespace URI
273  * enclosed in curly braces ({}), followed by the local name. If the
274  * name has a null URL, the String only contain the local name. An
275  * application can safely check for a non-null URI by testing to see if the
276  * first character of the name is a '{' character.</p>
277  * <p>For example, if a URI and local name were obtained from an element
278  * defined with <xyz:foo
279  * xmlns:xyz="http://xyz.foo.com/yada/baz.html">,
280  * then the qualified name would be "{http://xyz.foo.com/yada/baz.html}foo".
281  * Note that no prefix is used.</p>
282  *
283  * <p>The Properties object that was passed to {@link #setOutputProperties}
284  * won't be effected by calling this method.</p>
285  *
286  * @param name A non-null String that specifies an output
287  * property name, which may be namespace qualified.
288  * @param value The non-null string value of the output property.
289  *
290  * @throws IllegalArgumentException If the property is not supported, and is
291  * not qualified with a namespace.
292  *
293  * @see javax.xml.transform.OutputKeys
294  */
295 public abstract void setOutputProperty(String name, String value)
296     throws IllegalArgumentException;
297
298 /**
299  * <p>Get an output property that is in effect for the transformer.</p>
300  *
301  * <p>If a property has been set using {@link #setOutputProperty},
302  * that value will be returned. Otherwise, if a property is explicitly
303  * specified in the stylesheet, that value will be returned. If
304  * the value of the property has been defaulted, that is, if no
305  * value has been set explicitly either with {@link #setOutputProperty} or
306  * in the stylesheet, the result may vary depending on
307  * implementation and input stylesheet.</p>
308  *
309  * @param name A non-null String that specifies an output
310  * property name, which may be namespace qualified.
311  *
312  * @return The string value of the output property, or null
313  * if no property was found.
314  *
315  * @throws IllegalArgumentException If the property is not supported.
316  *
317  * @see javax.xml.transform.OutputKeys
318  */

```



```

319     public abstract String getOutputProperty(String name)
320         throws IllegalArgumentException;
321
322     /**
323      * Set the error event listener in effect for the transformation.
324      *
325      * @param listener The new error listener.
326      *
327      * @throws IllegalArgumentException if listener is null.
328      */
329     public abstract void setErrorListener(ErrorListener listener)
330         throws IllegalArgumentException;
331
332     /**
333      * Get the error event handler in effect for the transformation.
334      * Implementations must provide a default error listener.
335      *
336      * @return The current error handler, which should never be null.
337      */
338     public abstract ErrorListener getErrorListener();
339 }

```

## 21.10 TransformerConfigurationException.java

Secuencia 234: javax/xml/transform/TransformerConfigurationException.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 /**

```

```
29  * Indicates a serious configuration error.
30  */
31  public class TransformerConfigurationException extends TransformerException {
32
33      /**
34       * Create a new <code>TransformerConfigurationException</code> with no
35       * detail message.
36       */
37      public TransformerConfigurationException() {
38          super("Configuration Error");
39      }
40
41      /**
42       * Create a new <code>TransformerConfigurationException</code> with
43       * the <code>String</code> specified as an error message.
44       *
45       * @param msg The error message for the exception.
46       */
47      public TransformerConfigurationException(String msg) {
48          super(msg);
49      }
50
51      /**
52       * Create a new <code>TransformerConfigurationException</code> with a
53       * given <code>Exception</code> base cause of the error.
54       *
55       * @param e The exception to be encapsulated in a
56       * TransformerConfigurationException.
57       */
58      public TransformerConfigurationException(Throwable e) {
59          super(e);
60      }
61
62      /**
63       * Create a new <code>TransformerConfigurationException</code> with the
64       * given <code>Exception</code> base cause and detail message.
65       *
66       * @param e The exception to be encapsulated in a
67       * TransformerConfigurationException
68       * @param msg The detail message.
69       */
70      public TransformerConfigurationException(String msg, Throwable e) {
71          super(msg, e);
72      }
73
74      /**
75       * Create a new TransformerConfigurationException from a message and a Locator.
76       *
77       * <p>This constructor is especially useful when an application is
78       * creating its own exception from within a DocumentHandler
79       * callback.</p>
80       *
81       * @param message The error or warning message.
```

```

82     * @param locator The locator object for the error or warning.
83     */
84     public TransformerConfigurationException(String message,
85                                           SourceLocator locator) {
86         super(message, locator);
87     }
88
89     /**
90     * Wrap an existing exception in a TransformerConfigurationException.
91     *
92     * @param message The error or warning message, or null to
93     *                use the message from the embedded exception.
94     * @param locator The locator object for the error or warning.
95     * @param e Any exception.
96     */
97     public TransformerConfigurationException(String message,
98                                           SourceLocator locator,
99                                           Throwable e) {
100         super(message, locator, e);
101     }
102 }

```

## 21.11 TransformerException.java

Secuencia 235: javax/xml/transform/TransformerException.java

```

1  /*
2  * Copyright (c) 2000, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 import java.lang.reflect.Method;

```

```
29 import java.lang.reflect.InvocationTargetException;
30
31 /**
32  * This class specifies an exceptional condition that occurred
33  * during the transformation process.
34  */
35 public class TransformerException extends Exception {
36
37     /** Field locator specifies where the error occurred */
38     SourceLocator locator;
39
40     /**
41      * Method getLocator retrieves an instance of a SourceLocator
42      * object that specifies where an error occurred.
43      *
44      * @return A SourceLocator object, or null if none was specified.
45      */
46     public SourceLocator getLocator() {
47         return locator;
48     }
49
50     /**
51      * Method setLocator sets an instance of a SourceLocator
52      * object that specifies where an error occurred.
53      *
54      * @param location A SourceLocator object, or null to clear the location.
55      */
56     public void setLocator(SourceLocator location) {
57         locator = location;
58     }
59
60     /** Field containedException specifies a wrapped exception. May be null. */
61     Throwable containedException;
62
63     /**
64      * This method retrieves an exception that this exception wraps.
65      *
66      * @return An Throwable object, or null.
67      * @see #getCause
68      */
69     public Throwable getException() {
70         return containedException;
71     }
72
73     /**
74      * Returns the cause of this throwable or <code>null</code> if the
75      * cause is nonexistent or unknown. (The cause is the throwable that
76      * caused this throwable to get thrown.)
77      */
78     public Throwable getCause() {
79
80         return ((containedException == this)
81             ? null
```

```

82         : containedException);
83     }
84
85     /**
86      * Initializes the <i>cause</i> of this throwable to the specified value.
87      * (The cause is the throwable that caused this throwable to get thrown.)
88      *
89      * <p>This method can be called at most once. It is generally called from
90      * within the constructor, or immediately after creating the
91      * throwable. If this throwable was created
92      * with {@link #TransformerException(Throwable)} or
93      * {@link #TransformerException(String,Throwable)}, this method cannot be called
94      * even once.
95      *
96      * @param cause the cause (which is saved for later retrieval by the
97      *      {@link #getCause()} method). (A null value is
98      *      permitted, and indicates that the cause is nonexistent or
99      *      unknown.)
100     * @return a reference to this Throwable instance.
101     * @throws IllegalArgumentException if cause is this
102     *      throwable. (A throwable cannot
103     *      be its own cause.)
104     * @throws IllegalStateException if this throwable was
105     *      created with {@link #TransformerException(Throwable)} or
106     *      {@link #TransformerException(String,Throwable)}, or this method has already
107     *      been called on this throwable.
108     */
109     public synchronized Throwable initCause(Throwable cause) {
110
111         if (this.containedException != null) {
112             throw new IllegalStateException("Can't overwrite cause");
113         }
114
115         if (cause == this) {
116             throw new IllegalArgumentException(
117                 "Self-causation not permitted");
118         }
119
120         this.containedException = cause;
121
122         return this;
123     }
124
125     /**
126      * Create a new TransformerException.
127      *
128      * @param message The error or warning message.
129      */
130     public TransformerException(String message) {
131
132         super(message);
133
134         this.containedException = null;

```

```
135     this.locator          = null;
136 }
137
138 /**
139  * Create a new TransformerException wrapping an existing exception.
140  *
141  * @param e The exception to be wrapped.
142  */
143 public TransformerException(Throwable e) {
144
145     super(e.toString());
146
147     this.containedException = e;
148     this.locator           = null;
149 }
150
151 /**
152  * Wrap an existing exception in a TransformerException.
153  *
154  * <p>This is used for throwing processor exceptions before
155  * the processing has started.</p>
156  *
157  * @param message The error or warning message, or null to
158  *                use the message from the embedded exception.
159  * @param e Any exception
160  */
161 public TransformerException(String message, Throwable e) {
162
163     super(((message == null) || (message.length() == 0))
164           ? e.toString()
165           : message);
166
167     this.containedException = e;
168     this.locator           = null;
169 }
170
171 /**
172  * Create a new TransformerException from a message and a Locator.
173  *
174  * <p>This constructor is especially useful when an application is
175  * creating its own exception from within a DocumentHandler
176  * callback.</p>
177  *
178  * @param message The error or warning message.
179  * @param locator The locator object for the error or warning.
180  */
181 public TransformerException(String message, SourceLocator locator) {
182
183     super(message);
184
185     this.containedException = null;
186     this.locator           = locator;
187 }
```

```
188
189 /**
190  * Wrap an existing exception in a TransformerException.
191  *
192  * @param message The error or warning message, or null to
193  *                use the message from the embedded exception.
194  * @param locator The locator object for the error or warning.
195  * @param e Any exception
196  */
197 public TransformerException(String message, SourceLocator locator,
198                             Throwable e) {
199
200     super(message);
201
202     this.containedException = e;
203     this.locator            = locator;
204 }
205
206 /**
207  * Get the error message with location information
208  * appended.
209  *
210  * @return A <code>String</code> representing the error message with
211  *         location information appended.
212  */
213 public String getMessageAndLocation() {
214
215     StringBuffer sbuffer = new StringBuffer();
216     String        message = super.getMessage();
217
218     if (null != message) {
219         sbuffer.append(message);
220     }
221
222     if (null != locator) {
223         String systemID = locator.getSystemId();
224         int    line     = locator.getLineNumber();
225         int    column   = locator.getColumnNumber();
226
227         if (null != systemID) {
228             sbuffer.append("; SystemID: ");
229             sbuffer.append(systemID);
230         }
231
232         if (0 != line) {
233             sbuffer.append("; Line#: ");
234             sbuffer.append(line);
235         }
236
237         if (0 != column) {
238             sbuffer.append("; Column#: ");
239             sbuffer.append(column);
240         }
241     }
```

```
241     }
242
243     return sbuffer.toString();
244 }
245
246 /**
247  * Get the location information as a string.
248  *
249  * @return A string with location info, or null
250  * if there is no location information.
251  */
252 public String getLocationAsString() {
253
254     if (null != locator) {
255         StringBuffer sbuffer = new StringBuffer();
256         String      systemID = locator.getSystemId();
257         int         line    = locator.getLineNumber();
258         int         column   = locator.getColumnNumber();
259
260         if (null != systemID) {
261             sbuffer.append("; SystemID: ");
262             sbuffer.append(systemID);
263         }
264
265         if (0 != line) {
266             sbuffer.append("; Line#: ");
267             sbuffer.append(line);
268         }
269
270         if (0 != column) {
271             sbuffer.append("; Column#: ");
272             sbuffer.append(column);
273         }
274
275         return sbuffer.toString();
276     } else {
277         return null;
278     }
279 }
280
281 /**
282  * Print the the trace of methods from where the error
283  * originated. This will trace all nested exception
284  * objects, as well as this object.
285  */
286 public void printStackTrace() {
287     printStackTrace(new java.io.PrintWriter(System.err, true));
288 }
289
290 /**
291  * Print the the trace of methods from where the error
292  * originated. This will trace all nested exception
293  * objects, as well as this object.
```



```
294     * @param s The stream where the dump will be sent to.
295     */
296     public void printStackTrace(java.io.PrintStream s) {
297         printStackTrace(new java.io.PrintWriter(s));
298     }
299
300     /**
301     * Print the the trace of methods from where the error
302     * originated. This will trace all nested exception
303     * objects, as well as this object.
304     * @param s The writer where the dump will be sent to.
305     */
306     public void printStackTrace(java.io.PrintWriter s) {
307
308         if (s == null) {
309             s = new java.io.PrintWriter(System.err, true);
310         }
311
312         try {
313             String locInfo = getLocationAsString();
314
315             if (null != locInfo) {
316                 s.println(locInfo);
317             }
318
319             super.printStackTrace(s);
320         } catch (Throwable e) {}
321
322         Throwable exception = getException();
323
324         for (int i = 0; (i < 10) && (null != exception); i++) {
325             s.println("-----");
326
327             try {
328                 if (exception instanceof TransformerException) {
329                     String locInfo =
330                         ((TransformerException) exception)
331                             .getLocationAsString();
332
333                     if (null != locInfo) {
334                         s.println(locInfo);
335                     }
336                 }
337
338                 exception.printStackTrace(s);
339             } catch (Throwable e) {
340                 s.println("Could not print stack trace...");
341             }
342
343             try {
344                 Method meth =
345                     ((Object) exception).getClass().getMethod("getException",
346                         (Class[]) null);
```

```

347
348         if (null != meth) {
349             Throwable prev = exception;
350
351             exception = (Throwable) meth.invoke(exception, (Object[]) null);
352
353             if (prev == exception) {
354                 break;
355             }
356         } else {
357             exception = null;
358         }
359     } catch (InvocationTargetException ite) {
360         exception = null;
361     } catch (IllegalAccessException iae) {
362         exception = null;
363     } catch (NoSuchMethodException nsme) {
364         exception = null;
365     }
366 }
367 // insure output is written
368 s.flush();
369 }
370 }

```

## 21.12 TransformerFactory.java

Secuencia 236: javax/xml/transform/TransformerFactory.java

```

1  /*
2  * Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25

```

```
26 package javax.xml.transform;
27
28 /**
29  * <p>A TransformerFactory instance can be used to create
30  * {@link javax.xml.transform.Transformer} and
31  * {@link javax.xml.transform.Templates} objects.</p>
32  *
33  * <p>The system property that determines which Factory implementation
34  * to create is named <code>"javax.xml.transform.TransformerFactory"</code>.
35  * This property names a concrete subclass of the
36  * <code>TransformerFactory</code> abstract class. If the property is not
37  * defined, a platform default is be used.</p>
38  *
39  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
40  * @author <a href="mailto:Neeraj.Bajaj@sun.com">Neeraj Bajaj</a>
41  *
42  * @since 1.5
43  */
44 public abstract class TransformerFactory {
45
46     /**
47      * Default constructor is protected on purpose.
48      */
49     protected TransformerFactory() { }
50
51
52
53     /**
54      * <p>Obtain a new instance of a <code>TransformerFactory</code>.
55      * This static method creates a new factory instance.</p>
56      * <p>This method uses the following ordered lookup procedure to determine
57      * the <code>TransformerFactory</code> implementation class to
58      * load:</p>
59      * <ul>
60      * <li>
61      * Use the <code>javax.xml.transform.TransformerFactory</code> system
62      * property.
63      * </li>
64      * <li>
65      * Use the properties file "lib/jaxp.properties" in the JRE directory.
66      * This configuration file is in standard <code>java.util.Properties
67      * </code> format and contains the fully qualified name of the
68      * implementation class with the key being the system property defined
69      * above.
70      * <br>
71      * The jaxp.properties file is read only once by the JAXP implementation
72      * and it's values are then cached for future use. If the file does not exist
73      * when the first attempt is made to read from it, no further attempts are
74      * made to check for its existence. It is not possible to change the value
75      * of any property in jaxp.properties after it has been read for the first time.
76      * </li>
77      * <li>
78      * Use the service-provider loading facilities, defined by the
```

```

79      * {@link java.util.ServiceLoader} class, to attempt to locate and load an
80      * implementation of the service using the {@linkplain
81      * java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
82      * the service-provider loading facility will use the {@linkplain
83      * java.lang.Thread#getContextClassLoader() current thread's context class loader}
84      * to attempt to load the service. If the context class
85      * loader is null, the {@linkplain
86      * ClassLoader#getSystemClassLoader() system class loader} will be used.
87      * </li>
88      * <li>
89      * Otherwise, the system-default implementation is returned.
90      * </li>
91      * </ul>
92      *
93      * <p>Once an application has obtained a reference to a <code>
94      * TransformerFactory</code> it can use the factory to configure
95      * and obtain transformer instances.</p>
96      *
97      * @return new TransformerFactory instance, never null.
98      *
99      * @throws TransformerFactoryConfigurationError Thrown in case of {@linkplain
100     * java.util.ServiceConfigurationError service configuration error} or if
101     * the implementation is not available or cannot be instantiated.
102     */
103     public static TransformerFactory newInstance()
104         throws TransformerFactoryConfigurationError {
105
106         return FactoryFinder.find(
107             /* The default property name according to the JAXP spec */
108             TransformerFactory.class,
109             /* The fallback implementation class name, XSLTC */
110             "com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl");
111     }
112
113     /**
114     * <p>Obtain a new instance of a <code>TransformerFactory</code> from factory class name.
115     * This function is useful when there are multiple providers in the classpath.
116     * It gives more control to the application as it can specify which provider
117     * should be loaded.</p>
118     *
119     * <p>Once an application has obtained a reference to a <code>
120     * TransformerFactory</code> it can use the factory to configure
121     * and obtain transformer instances.</p>
122     *
123     * <h2>Tip for Trouble-shooting</h2>
124     * <p>Setting the <code>jaxp.debug</code> system property will cause
125     * this method to print a lot of debug messages
126     * to <code>System.err</code> about what it is doing and where it is looking at.</p>
127     *
128     * <p> If you have problems try:</p>
129     * <pre>
130     * java -Djaxp.debug=1 YourProgram ....
131     * </pre>

```

```

132     *
133     * @param factoryClassName fully qualified factory class name that provides implementation of <
134     *     code>javax.xml.transform.TransformerFactory</code>.
135     *
136     * @param classLoader <code>ClassLoader</code> used to load the factory class. If <code>null</
137     *     code>
138     *         current <code>Thread</code>'s context classLoader is used to load the
139     *         factory class.
140     *
141     * @return new TransformerFactory instance, never null.
142     *
143     * @throws TransformerFactoryConfigurationException
144     *     if <code>factoryClassName</code> is <code>null</code>, or
145     *     the factory class cannot be loaded, instantiated.
146     *
147     * @see #newInstance()
148     *
149     * @since 1.6
150     */
151     public static TransformerFactory newInstance(String factoryClassName, ClassLoader classLoader)
152         throws TransformerFactoryConfigurationException{
153
154         //do not fallback if given classloader can't find the class, throw exception
155         return FactoryFinder.newInstance(TransformerFactory.class,
156             factoryClassName, classLoader, false, false);
157     }
158     /**
159     * <p>Process the <code>Source</code> into a <code>Transformer</code>
160     * <code>Object</code>. The <code>Source</code> is an XSLT document that
161     * conforms to <a href="http://www.w3.org/TR/xslt">
162     * XSL Transformations (XSLT) Version 1.0</a>. Care must
163     * be taken not to use this <code>Transformer</code> in multiple
164     * <code>Thread</code>s running concurrently.
165     * Different <code>TransformerFactories</code> can be used concurrently by
166     * different <code>Thread</code>s.</p>
167     *
168     * @param source <code>Source</code> of XSLT document used to create
169     * <code>Transformer</code>.
170     * Examples of XML <code>Source</code>s include
171     * {<code>@link javax.xml.transform.dom.DOMSource DOMSource</code>},
172     * {<code>@link javax.xml.transform.sax.SAXSource SAXSource</code>}, and
173     * {<code>@link javax.xml.transform.stream.StreamSource StreamSource</code>}.
174     *
175     * @return A <code>Transformer</code> object that may be used to perform
176     * a transformation in a single <code>Thread</code>, never
177     * <code>null</code>.
178     *
179     * @throws TransformerConfigurationException Thrown if there are errors when
180     * parsing the <code>Source</code> or it is not possible to create a
181     * <code>Transformer</code> instance.
182     *
183     * @see <a href="http://www.w3.org/TR/xslt">
184     * XSL Transformations (XSLT) Version 1.0</a>

```

```

182     */
183     public abstract Transformer newTransformer(Source source)
184         throws TransformerConfigurationException;
185
186     /**
187     * <p>Create a new <code>Transformer</code> that performs a copy
188     * of the <code>Source</code> to the <code>Result</code>.
189     * i.e. the "<em>identity transform</em>".</p>
190     *
191     * @return A Transformer object that may be used to perform a transformation
192     * in a single thread, never null.
193     *
194     * @throws TransformerConfigurationException When it is not
195     * possible to create a <code>Transformer</code> instance.
196     */
197     public abstract Transformer newTransformer()
198         throws TransformerConfigurationException;
199
200     /**
201     * Process the Source into a Templates object, which is a
202     * a compiled representation of the source. This Templates object
203     * may then be used concurrently across multiple threads. Creating
204     * a Templates object allows the TransformerFactory to do detailed
205     * performance optimization of transformation instructions, without
206     * penalizing runtime transformation.
207     *
208     * @param source An object that holds a URL, input stream, etc.
209     *
210     * @return A Templates object capable of being used for transformation
211     * purposes, never <code>null</code>.
212     *
213     * @throws TransformerConfigurationException When parsing to
214     * construct the Templates object fails.
215     */
216     public abstract Templates newTemplates(Source source)
217         throws TransformerConfigurationException;
218
219     /**
220     * <p>Get the stylesheet specification(s) associated with the
221     * XML <code>Source</code> document via the
222     * <a href="http://www.w3.org/TR/xml-stylesheet/">
223     * xml-stylesheet processing instruction</a> that match the given criteria.
224     * Note that it is possible to return several stylesheets, in which case
225     * they are applied as if they were a list of imports or cascades in a
226     * single stylesheet.</p>
227     *
228     * @param source The XML source document.
229     * @param media The media attribute to be matched. May be null, in which
230     * case the preferred templates will be used (i.e. alternate = no).
231     * @param title The value of the title attribute to match. May be null.
232     * @param charset The value of the charset attribute to match. May be null.
233     *
234     * @return A <code>Source</code> <code>Object</code> suitable for passing

```

```

235     * to the TransformerFactory.
236     *
237     * @throws TransformerConfigurationException An Exception
238     * is thrown if an error occurs during parsing of the
239     * source.
240     *
241     * @see <a href="http://www.w3.org/TR/xml-stylesheet/">
242     * Associating Style Sheets with XML documents Version 1.0</a>
243     */
244     public abstract Source getAssociatedStylesheet(
245         Source source,
246         String media,
247         String title,
248         String charset)
249         throws TransformerConfigurationException;
250
251     /**
252     * Set an object that is used by default during the transformation
253     * to resolve URIs used in document(), xsl:import, or xsl:include.
254     *
255     * @param resolver An object that implements the URIResolver interface,
256     * or null.
257     */
258     public abstract void setURIResolver(URIResolver resolver);
259
260     /**
261     * Get the object that is used by default during the transformation
262     * to resolve URIs used in document(), xsl:import, or xsl:include.
263     *
264     * @return The URIResolver that was set with setURIResolver.
265     */
266     public abstract URIResolver getURIResolver();
267
268     //===== CONFIGURATION METHODS =====
269
270     /**
271     * <p>Set a feature for this TransformerFactory and Transformers
272     * or Templates created by this factory.</p>
273     *
274     * <p>
275     * Feature names are fully qualified {@link java.net.URI}s.
276     * Implementations may define their own features.
277     * An {@link TransformerConfigurationException} is thrown if this TransformerFactory
278     * or the Transformers or Templates it creates cannot support the
279     * feature.
280     * It is possible for an TransformerFactory to expose a feature value but be
281     * unable to change its state.
282     * </p>
283     *
284     * <p>All implementations are required to support the {@link javax.xml.XMLConstants#
285     * FEATURE_SECURE_PROCESSING} feature.
286     * When the feature is:</p>

```

```

284     * <ul>
285     *   <li>
286     *     <code>true</code>: the implementation will limit XML processing to conform to
287     *       implementation limits
288     *       and behave in a secure fashion as defined by the implementation.
289     *       Examples include resolving user defined style sheets and functions.
290     *       If XML processing is limited for security reasons, it will be reported via a call to
291     *       the registered
292     *       {@link ErrorListener#fatalError(TransformerException exception)}.
293     *       See {@link #setErrorListener(ErrorListener listener)}.
294     *   </li>
295     *   <li>
296     *     <code>false</code>: the implementation will processing XML according to the XML
297     *       specifications without
298     *       regard to possible implementation limits.
299     *   </li>
300     * </ul>
301     *
302     * @param name Feature name.
303     * @param value Is feature state <code>true</code> or <code>false</code>.
304     *
305     * @throws TransformerConfigurationException if this <code>TransformerFactory</code>
306     *   or the <code>Transformer</code>s or <code>Template</code>s it creates cannot support
307     *   this feature.
308     * @throws NullPointerException If the <code>name</code> parameter is null.
309     */
310     public abstract void setFeature(String name, boolean value)
311       throws TransformerConfigurationException;
312
313 /**
314  * Look up the value of a feature.
315  *
316  * <p>
317  * Feature names are fully qualified {@link java.net.URI}s.
318  * Implementations may define their own features.
319  * <code>false</code> is returned if this <code>TransformerFactory</code> or the
320  * <code>Transformer</code>s or <code>Template</code>s it creates cannot support the
321  * feature.
322  * It is possible for an <code>TransformerFactory</code> to expose a feature value but be
323  * unable to change its state.
324  * </p>
325  *
326  * @param name Feature name.
327  *
328  * @return The current state of the feature, <code>true</code> or <code>false</code>.
329  *
330  * @throws NullPointerException If the <code>name</code> parameter is null.
331  */
332     public abstract boolean getFeature(String name);
333
334 /**
335  * Allows the user to set specific attributes on the underlying
336  * implementation. An attribute in this context is defined to

```



```

331     * be an option that the implementation provides.
332     * An IllegalArgumentException is thrown if the underlying
333     * implementation doesn't recognize the attribute.
334     * <p>
335     * All implementations that implement JAXP 1.5 or newer are required to
336     * support the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} and
337     * {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_STYLESHEET} properties.
338     * </p>
339     * <ul>
340     *   <li>
341     *     <p>
342     *       Access to external DTDs in the source file is restricted to the protocols
343     *       specified by the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} property.
344     *       If access is denied during transformation due to the restriction of this property,
345     *       {@link javax.xml.transform.TransformerException} will be thrown by
346     *       {@link javax.xml.transform.Transformer#transform(Source, Result)}.
347     *     </p>
348     *     <p>
349     *       Access to external DTDs in the stylesheet is restricted to the protocols
350     *       specified by the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} property.
351     *       If access is denied during the creation of a new transformer due to the
352     *       restriction of this property,
353     *       {@link javax.xml.transform.TransformerConfigurationException} will be thrown
354     *       by the {@link #newTransformer(Source)} method.
355     *     </p>
356     *     <p>
357     *       Access to external reference set by the stylesheet processing instruction,
358     *       Import and Include element is restricted to the protocols specified by the
359     *       {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_STYLESHEET} property.
360     *       If access is denied during the creation of a new transformer due to the
361     *       restriction of this property,
362     *       {@link javax.xml.transform.TransformerConfigurationException} will be thrown
363     *       by the {@link #newTransformer(Source)} method.
364     *     </p>
365     *     <p>
366     *       Access to external document through XSLT document function is restricted
367     *       to the protocols specified by the property. If access is denied during
368     *       the transformation due to the restriction of this property,
369     *       {@link javax.xml.transform.TransformerException} will be thrown by the
370     *       {@link javax.xml.transform.Transformer#transform(Source, Result)} method.
371     *     </p>
372     *   </li>
373     * </ul>
374     *
375     * @param name The name of the attribute.
376     * @param value The value of the attribute.
377     *
378     * @throws IllegalArgumentException When implementation does not
379     *   recognize the attribute.
380     */
381     public abstract void setAttribute(String name, Object value);
382
383     /**

```

```

384     * Allows the user to retrieve specific attributes on the underlying
385     * implementation.
386     * An <code>IllegalArgumentException</code> is thrown if the underlying
387     * implementation doesn't recognize the attribute.
388     *
389     * @param name The name of the attribute.
390     *
391     * @return value The value of the attribute.
392     *
393     * @throws IllegalArgumentException When implementation does not
394     *     recognize the attribute.
395     */
396     public abstract Object getAttribute(String name);
397
398     /**
399     * Set the error event listener for the TransformerFactory, which
400     * is used for the processing of transformation instructions,
401     * and not for the transformation itself.
402     * An <code>IllegalArgumentException</code> is thrown if the
403     * <code>ErrorListener</code> listener is <code>null</code>.
404     *
405     * @param listener The new error listener.
406     *
407     * @throws IllegalArgumentException When <code>listener</code> is
408     *     <code>null</code>
409     */
410     public abstract void setErrorListener(ErrorListener listener);
411
412     /**
413     * Get the error event handler for the TransformerFactory.
414     *
415     * @return The current error handler, which should never be null.
416     */
417     public abstract ErrorListener getErrorListener();
418
419 }

```

## 21.13 TransformerFactoryConfigurationError.java

Secuencia 237: javax/xml/transform/TransformerFactoryConfigurationError.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.transform;
27
28 /**
29  * Thrown when a problem with configuration with the Transformer Factories
30  * exists. This error will typically be thrown when the class of a
31  * transformation factory specified in the system properties cannot be found
32  * or instantiated.
33  */
34 public class TransformerFactoryConfigurationError extends Error {
35     private static final long serialVersionUID = -6527718720676281516L;
36
37     /**
38      * <code>Exception</code> for the
39      * <code>TransformerFactoryConfigurationError</code>.
40      */
41     private Exception exception;
42
43     /**
44      * Create a new <code>TransformerFactoryConfigurationError</code> with no
45      * detail message.
46      */
47     public TransformerFactoryConfigurationError() {
48
49         super();
50
51         this.exception = null;
52     }
53
54     /**
55      * Create a new <code>TransformerFactoryConfigurationError</code> with
56      * the <code>String</code> specified as an error message.
57      *
58      * @param msg The error message for the exception.
59      */
60     public TransformerFactoryConfigurationError(String msg) {
61
62         super(msg);
63
64         this.exception = null;
65     }
66 }
```

```
67  /**
68   * Create a new TransformerFactoryConfigurationError with a
69   * given Exception base cause of the error.
70   *
71   * @param e The exception to be encapsulated in a
72   * TransformerFactoryConfigurationError.
73   */
74  public TransformerFactoryConfigurationError(Exception e) {
75
76      super(e.toString());
77
78      this.exception = e;
79  }
80
81  /**
82   * Create a new TransformerFactoryConfigurationError with the
83   * given Exception base cause and detail message.
84   *
85   * @param e The exception to be encapsulated in a
86   * TransformerFactoryConfigurationError
87   * @param msg The detail message.
88   */
89  public TransformerFactoryConfigurationError(Exception e, String msg) {
90
91      super(msg);
92
93      this.exception = e;
94  }
95
96  /**
97   * Return the message (if any) for this error . If there is no
98   * message for the exception and there is an encapsulated
99   * exception then the message of that exception will be returned.
100  *
101  * @return The error message.
102  */
103  public String getMessage() {
104
105      String message = super.getMessage();
106
107      if ((message == null) && (exception != null)) {
108          return exception.getMessage();
109      }
110
111      return message;
112  }
113
114  /**
115   * Return the actual exception (if any) that caused this exception to
116   * be raised.
117   *
118   * @return The encapsulated exception, or null if there is none.
119   */
```

```

120     public Exception getException() {
121         return exception;
122     }
123     /**
124      * use the exception chaining mechanism of JDK1.4
125      */
126     @Override
127     public Throwable getCause() {
128         return exception;
129     }
130 }

```

## 21.14 URIResolver.java

Secuencia 238: javax.xml.transform/URIResolver.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform;
27
28 /**
29  * <p>An object that implements this interface that can be called by the processor
30  * to turn a URI used in document(), xsl:import, or xsl:include into a Source object.
31  */
32 public interface URIResolver {
33
34     /**
35      * Called by the processor when it encounters
36      * an xsl:include, xsl:import, or document() function.
37      *
38      * @param href An href attribute, which may be relative or absolute.

```

```

39     * @param base The base URI against which the first argument will be made
40     * absolute if the absolute URI is required.
41     *
42     * @return A Source object, or null if the href cannot be resolved,
43     * and the processor should try to resolve the URI itself.
44     *
45     * @throws TransformerException if an error occurs when trying to
46     * resolve the URI.
47     */
48     public Source resolve(String href, String base)
49         throws TransformerException;
50 }

```

## 22 Xml Transform Dom (javax.xml.transform.dom package)

### 22.1 DOMLocator.java

Secuencia 239: javax.xml.transform.dom/DOMLocator.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.dom;
27
28 import javax.xml.transform.SourceLocator;
29
30 import org.w3c.dom.Node;
31
32
33 /**
34  * Indicates the position of a node in a source DOM, intended
35  * primarily for error reporting. To use a DOMLocator, the receiver of an

```

```

36 * error must downcast the {@link javax.xml.transform.SourceLocator}
37 * object returned by an exception. A {@link javax.xml.transform.Transformer}
38 * may use this object for purposes other than error reporting, for instance,
39 * to indicate the source node that originated a result node.
40 */
41 public interface DOMLocator extends SourceLocator {
42
43     /**
44      * Return the node where the event occurred.
45      *
46      * @return The node that is the location for the event.
47      */
48     public Node getOriginatingNode();
49 }

```

## 22.2 DOMResult.java

Secuencia 240: javax/xml/transform/dom/DOMResult.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.dom;
27
28 import javax.xml.transform.Result;
29 import org.w3c.dom.Node;
30
31 /**
32 * <p>Acts as a holder for a transformation result tree in the form of a Document Object Model (DOM
33 *   ) tree.</p>
34 *

```

```

34 * <p>If no output DOM source is set, the transformation will create a Document node as the holder
    for the result of the transformation,
35 * which may be retrieved with {@link #getNode()}.</p>
36 *
37 * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>
38 */
39 public class DOMResult implements Result {
40
41     /** <p>If {@link javax.xml.transform.TransformerFactory#getFeature}
42      * returns <code>>true</code> when passed this value as an argument,
43      * the <code>Transformer</code> supports <code>Result</code> output of this type.</p>
44      */
45     public static final String FEATURE = "http://javax.xml.transform.dom.DOMResult/feature";
46
47     /**
48      * <p>Zero-argument default constructor.</p>
49      *
50      * <p><code>node</code>,
51      * <code>siblingNode</code> and
52      * <code>systemId</code>
53      * will be set to <code>null</code>.</p>
54      */
55     public DOMResult() {
56         setNode(null);
57         setNextSibling(null);
58         setSystemId(null);
59     }
60
61     /**
62      * <p>Use a DOM node to create a new output target.</p>
63      *
64      * <p>In practice, the node should be
65      * a {@link org.w3c.dom.Document} node,
66      * a {@link org.w3c.dom.DocumentFragment} node, or
67      * a {@link org.w3c.dom.Element} node.
68      * In other words, a node that accepts children.</p>
69      *
70      * <p><code>siblingNode</code> and
71      * <code>systemId</code>
72      * will be set to <code>null</code>.</p>
73      *
74      * @param node The DOM node that will contain the result tree.
75      */
76     public DOMResult(Node node) {
77         setNode(node);
78         setNextSibling(null);
79         setSystemId(null);
80     }
81
82     /**
83      * <p>Use a DOM node to create a new output target with the specified System ID.<p>
84      *
85      * <p>In practice, the node should be

```



```

86     * a {@link org.w3c.dom.Document} node,
87     * a {@link org.w3c.dom.DocumentFragment} node, or
88     * a {@link org.w3c.dom.Element} node.
89     * In other words, a node that accepts children.</p>
90     *
91     * <p><code>siblingNode</code> will be set to <code>null</code>.</p>
92     *
93     * @param node The DOM node that will contain the result tree.
94     * @param systemId The system identifier which may be used in association with this node.
95     */
96     public DOMResult(Node node, String systemId) {
97         setNode(node);
98         setNextSibling(null);
99         setSystemId(systemId);
100    }
101
102    /**
103     * <p>Use a DOM node to create a new output target specifying the child node where the result
104     * nodes should be inserted before.</p>
105     *
106     * <p>In practice, <code>node</code> and <code>nextSibling</code> should be
107     * a {@link org.w3c.dom.Document} node,
108     * a {@link org.w3c.dom.DocumentFragment} node, or
109     * a {@link org.w3c.dom.Element} node.
110     * In other words, a node that accepts children.</p>
111     *
112     * <p>Use <code>nextSibling</code> to specify the child node
113     * where the result nodes should be inserted before.
114     * If <code>nextSibling</code> is not a sibling of <code>node</code>,
115     * then an <code>IllegalArgumentException</code> is thrown.
116     * If <code>node</code> is <code>null</code> and <code>nextSibling</code> is not <code>null</code>,
117     * then an <code>IllegalArgumentException</code> is thrown.
118     * If <code>nextSibling</code> is <code>null</code>,
119     * then the behavior is the same as calling {@link #DOMResult(Node node)},
120     * i.e. append the result nodes as the last child of the specified <code>node</code>.</p>
121     *
122     * <p><code>systemId</code> will be set to <code>null</code>.</p>
123     *
124     * @param node The DOM node that will contain the result tree.
125     * @param nextSibling The child node where the result nodes should be inserted before.
126     *
127     * @throws IllegalArgumentException If <code>nextSibling</code> is not a sibling of <code>node</code> or
128     * <code>node</code> is <code>null</code> and <code>nextSibling</code>
129     * is not <code>null</code>.
130     *
131     * @since 1.5
132     */
133     public DOMResult(Node node, Node nextSibling) {
134         // does the current parent/child relationship exist?
135         if (nextSibling != null) {

```

```

136         // cannot be a sibling of a null node
137         if (node == null) {
138             throw new IllegalArgumentException("Cannot create a DOMResult when the nextSibling
139                 is contained by the \"null\" node.");
140         }
141
142         // nextSibling contained by node?
143         if ((node.compareDocumentPosition(nextSibling)&Node.DOCUMENT_POSITION_CONTAINED_BY)==0)
144             {
145                 throw new IllegalArgumentException("Cannot create a DOMResult when the nextSibling
146                     is not contained by the node.");
147             }
148         }
149
150         setNode(node);
151         setNextSibling(nextSibling);
152         setSystemId(null);
153     }
154
155     /**
156     * <p>Use a DOM node to create a new output target specifying the child node where the result
157     * nodes should be inserted before and
158     * the specified System ID.</p>
159     *
160     * <p>In practice, <code>node</code> and <code>nextSibling</code> should be
161     * a {@link org.w3c.dom.Document} node,
162     * a {@link org.w3c.dom.DocumentFragment} node, or a
163     * {@link org.w3c.dom.Element} node.
164     * In other words, a node that accepts children.</p>
165     *
166     * <p>Use <code>nextSibling</code> to specify the child node
167     * where the result nodes should be inserted before.
168     * If <code>nextSibling</code> is not a sibling of <code>node</code>,
169     * then an <code>IllegalArgumentException</code> is thrown.
170     * If <code>node</code> is <code>null</code> and <code>nextSibling</code> is not <code>null</code>,
171     * then an <code>IllegalArgumentException</code> is thrown.
172     * If <code>nextSibling</code> is <code>null</code>,
173     * then the behavior is the same as calling {@link #DOMResult(Node node, String systemId)},
174     * i.e. append the result nodes as the last child of the specified node and use the specified
175     * System ID.</p>
176     *
177     * @param node The DOM node that will contain the result tree.
178     * @param nextSibling The child node where the result nodes should be inserted before.
179     * @param systemId The system identifier which may be used in association with this node.
180     *
181     * @throws IllegalArgumentException If <code>nextSibling</code> is not a
182     *     sibling of <code>node</code> or
183     *     <code>node</code> is <code>null</code> and <code>nextSibling</code>
184     *     is not <code>null</code>.
185     *
186     * @since 1.5
187     */

```

```

183 public DOMResult(Node node, Node nextSibling, String systemId) {
184
185     // does the corrent parent/child relationship exist?
186     if (nextSibling != null) {
187         // cannot be a sibling of a null node
188         if (node == null) {
189             throw new IllegalArgumentException("Cannot create a DOMResult when the nextSibling
190                 is contained by the \"null\" node.");
191         }
192
193         // nextSibling contained by node?
194         if ((node.compareDocumentPosition(nextSibling)&Node.DOCUMENT_POSITION_CONTAINED_BY)==0)
195             {
196                 throw new IllegalArgumentException("Cannot create a DOMResult when the nextSibling
197                     is not contained by the node.");
198             }
199     }
200
201     setNode(node);
202     setNextSibling(nextSibling);
203     setSystemId(systemId);
204 }
205
206 /**
207  * <p>Set the node that will contain the result DOM tree.<p>
208  *
209  * <p>In practice, the node should be
210  * a {@link org.w3c.dom.Document} node,
211  * a {@link org.w3c.dom.DocumentFragment} node, or
212  * a {@link org.w3c.dom.Element} node.
213  * In other words, a node that accepts children.</p>
214  *
215  * <p>An IllegalStateException is thrown if
216  * nextSibling is not null and
217  * node is not a parent of nextSibling.
218  * An IllegalStateException is thrown if node is null
219  * and
220  * nextSibling is not null.</p>
221  *
222  * @param node The node to which the transformation will be appended.
223  *
224  * @throws IllegalStateException If nextSibling is not
225  * null and
226  * nextSibling is not a child of node or
227  * node is null and
228  * nextSibling is not null.
229  */
230 public void setNode(Node node) {
231     // does the corrent parent/child relationship exist?
232     if (nextSibling != null) {
233         // cannot be a sibling of a null node
234         if (node == null) {
235             throw new IllegalStateException("Cannot create a DOMResult when the nextSibling is

```

```

        contained by the \"null\" node.");
232     }
233
234     // nextSibling contained by node?
235     if ((node.compareDocumentPosition(nextSibling)&Node.DOCUMENT_POSITION_CONTAINED_BY)==0)
        {
236         throw new IllegalArgumentException("Cannot create a DOMResult when the nextSibling
            is not contained by the node.");
237     }
238 }
239
240 this.node = node;
241 }
242
243 /**
244  * <p>Get the node that will contain the result DOM tree.</p>
245  *
246  * <p>If no node was set via
247  * {@link #DOMResult(Node node)},
248  * {@link #DOMResult(Node node, String sysId)},
249  * {@link #DOMResult(Node node, Node nextSibling)},
250  * {@link #DOMResult(Node node, Node nextSibling, String sysId)} or
251  * {@link #setNode(Node node)},
252  * then the node will be set by the transformation, and may be obtained from this method once
        the transformation is complete.
253  * Calling this method before the transformation will return <code>null</code>.</p>
254  *
255  * @return The node to which the transformation will be appended.
256  */
257 public Node getNode() {
258     return node;
259 }
260
261 /**
262  * <p>Set the child node before which the result nodes will be inserted.</p>
263  *
264  * <p>Use <code>nextSibling</code> to specify the child node
265  * before which the result nodes should be inserted.
266  * If <code>nextSibling</code> is not a descendant of <code>node</code>,
267  * then an <code>IllegalArgumentException</code> is thrown.
268  * If <code>node</code> is <code>null</code> and <code>nextSibling</code> is not <code>null</code>
        <code>,
269  * then an <code>IllegalStateException</code> is thrown.
270  * If <code>nextSibling</code> is <code>null</code>,
271  * then the behavior is the same as calling {@link #DOMResult(Node node)},
272  * i.e. append the result nodes as the last child of the specified <code>node</code>.</p>
273  *
274  * @param nextSibling The child node before which the result nodes will be inserted.
275  *
276  * @throws IllegalArgumentException If <code>nextSibling</code> is not a
277  *     descendant of <code>node</code>.
278  * @throws IllegalStateException If <code>node</code> is <code>null</code>
279  *     and <code>nextSibling</code> is not <code>null</code>.

```

```

280     *
281     * @since 1.5
282     */
283     public void setNextSibling(Node nextSibling) {
284
285         // does the corrent parent/child relationship exist?
286         if (nextSibling != null) {
287             // cannot be a sibling of a null node
288             if (node == null) {
289                 throw new IllegalStateException("Cannot create a DOMResult when the nextSibling is
                contained by the \"null\" node.");
290             }
291
292             // nextSibling contained by node?
293             if ((node.compareDocumentPosition(nextSibling)&Node.DOCUMENT_POSITION_CONTAINED_BY)==0)
                {
294                 throw new IllegalArgumentException("Cannot create a DOMResult when the nextSibling
                is not contained by the node.");
295             }
296         }
297
298         this.nextSibling = nextSibling;
299     }
300
301     /**
302     * <p>Get the child node before which the result nodes will be inserted.</p>
303     *
304     * <p>If no node was set via
305     * {@link #DOMResult(Node node, Node nextSibling)},
306     * {@link #DOMResult(Node node, Node nextSibling, String systemId)} or
307     * {@link #setNextSibling(Node nextSibling)},
308     * then <code>null</code> will be returned.</p>
309     *
310     * @return The child node before which the result nodes will be inserted.
311     *
312     * @since 1.5
313     */
314     public Node getNextSibling() {
315         return nextSibling;
316     }
317
318     /**
319     * <p>Set the systemId that may be used in association with the node.</p>
320     *
321     * @param systemId The system identifier as a URI string.
322     */
323     public void setSystemId(String systemId) {
324         this.systemId = systemId;
325     }
326
327     /**
328     * <p>Get the System Identifier.</p>
329     *

```

```

330 * <p>If no System ID was set via
331 * {@link #DOMResult(Node node, String systemId)},
332 * {@link #DOMResult(Node node, Node nextSibling, String systemId)} or
333 * {@link #setSystemId(String systemId)},
334 * then <code>null</code> will be returned.</p>
335 *
336 * @return The system identifier.
337 */
338 public String getSystemId() {
339     return systemId;
340 }
341
342 ////////////////////////////////////////////////////
343 // Internal state.
344 ////////////////////////////////////////////////////
345
346 /**
347 * <p>The node to which the transformation will be appended.</p>
348 */
349 private Node node = null;
350
351 /**
352 * <p>The child node before which the result nodes will be inserted.</p>
353 *
354 * @since 1.5
355 */
356 private Node nextSibling = null;
357
358 /**
359 * <p>The System ID that may be used in association with the node.</p>
360 */
361 private String systemId = null;
362 }

```

## 22.3 DOMSource.java

Secuencia 241: javax/xml/transform/dom/DOMSource.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *

```

```
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.transform.dom;
27
28 import javax.xml.transform.Source;
29
30 import org.w3c.dom.Node;
31
32 /**
33  * <p>Acts as a holder for a transformation Source tree in the
34  * form of a Document Object Model (DOM) tree.</p>
35  *
36  * <p>Note that XSLT requires namespace support. Attempting to transform a DOM
37  * that was not constructed with a namespace-aware parser may result in errors.
38  * Parsers can be made namespace aware by calling
39  * {@link javax.xml.parsers.DocumentBuilderFactory#setNamespaceAware(boolean awareness)}.</p>
40  *
41  * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>
42  * @see <a href="http://www.w3.org/TR/DOM-Level-2">Document Object Model (DOM) Level 2
43  *     Specification</a>
44  */
45
46 public class DOMSource implements Source {
47
48     /**
49      * <p><code>Node</code> to serve as DOM source.</p>
50      */
51     private Node node;
52
53     /**
54      * <p>The base ID (URL or system ID) from where URLs
55      * will be resolved.</p>
56      */
57     private String systemID;
58
59     /** If {@link javax.xml.transform.TransformerFactory#getFeature}
60      * returns true when passed this value as an argument,
61      * the Transformer supports Source input of this type.
62      */
63     public static final String FEATURE =
64         "http://javax.xml.transform.dom.DOMSource/feature";
65
66     /**
67      * <p>Zero-argument default constructor. If this constructor is used, and
68      * no DOM source is set using {@link #setNode(Node node)} , then the
69      * <code>Transformer</code> will
70      * create an empty source {@link org.w3c.dom.Document} using
```

```

69      * {@link javax.xml.parsers.DocumentBuilder#newDocument()}.</p>
70      *
71      * @see javax.xml.transform.Transformer#transform(Source xmlSource, Result outputTarget)
72      */
73      public DOMSource() { }
74
75      /**
76       * Create a new input source with a DOM node. The operation
77       * will be applied to the subtree rooted at this node. In XSLT,
78       * a "/" pattern still means the root of the tree (not the subtree),
79       * and the evaluation of global variables and parameters is done
80       * from the root node also.
81       *
82       * @param n The DOM node that will contain the Source tree.
83       */
84      public DOMSource(Node n) {
85          setNode(n);
86      }
87
88      /**
89       * Create a new input source with a DOM node, and with the
90       * system ID also passed in as the base URI.
91       *
92       * @param node The DOM node that will contain the Source tree.
93       * @param systemID Specifies the base URI associated with node.
94       */
95      public DOMSource(Node node, String systemID) {
96          setNode(node);
97          setSystemId(systemID);
98      }
99
100     /**
101      * Set the node that will represents a Source DOM tree.
102      *
103      * @param node The node that is to be transformed.
104      */
105     public void setNode(Node node) {
106         this.node = node;
107     }
108
109     /**
110      * Get the node that represents a Source DOM tree.
111      *
112      * @return The node that is to be transformed.
113      */
114     public Node getNode() {
115         return node;
116     }
117
118     /**
119      * Set the base ID (URL or system ID) from where URLs
120      * will be resolved.
121      *

```



```

122     * @param systemID Base URL for this DOM tree.
123     */
124     public void setSystemId(String systemID) {
125         this.systemID = systemID;
126     }
127
128     /**
129     * Get the base ID (URL or system ID) from where URLs
130     * will be resolved.
131     *
132     * @return Base URL for this DOM tree.
133     */
134     public String getSystemId() {
135         return this.systemID;
136     }
137 }

```

## 23 Xml Transform Sax (javax.xml.transform.sax package)

### 23.1 SAXResult.java

Secuencia 242: javax.xml.transform.sax/SAXResult.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.sax;
27
28 import javax.xml.transform.Result;
29
30 import org.xml.sax.ContentHandler;
31 import org.xml.sax.ext.LexicalHandler;

```

```

32
33 /**
34  * <p>Acts as an holder for a transformation Result.</p>
35  *
36  * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>
37  */
38 public class SAXResult implements Result {
39
40     /**
41      * If {@link javax.xml.transform.TransformerFactory#getFeature}
42      * returns true when passed this value as an argument,
43      * the Transformer supports Result output of this type.
44      */
45     public static final String FEATURE =
46         "http://javax.xml.transform.sax.SAXResult/feature";
47
48     /**
49      * Zero-argument default constructor.
50      */
51     public SAXResult() {
52     }
53
54     /**
55      * Create a SAXResult that targets a SAX2 {@link org.xml.sax.ContentHandler}.
56      *
57      * @param handler Must be a non-null ContentHandler reference.
58      */
59     public SAXResult(ContentHandler handler) {
60         setHandler(handler);
61     }
62
63     /**
64      * Set the target to be a SAX2 {@link org.xml.sax.ContentHandler}.
65      *
66      * @param handler Must be a non-null ContentHandler reference.
67      */
68     public void setHandler(ContentHandler handler) {
69         this.handler = handler;
70     }
71
72     /**
73      * Get the {@link org.xml.sax.ContentHandler} that is the Result.
74      *
75      * @return The ContentHandler that is to be transformation output.
76      */
77     public ContentHandler getHandler() {
78         return handler;
79     }
80
81     /**
82      * Set the SAX2 {@link org.xml.sax.ext.LexicalHandler} for the output.
83      *
84      * <p>This is needed to handle XML comments and the like. If the

```

```

85     * lexical handler is not set, an attempt should be made by the
86     * transformer to cast the {@link org.xml.sax.ContentHandler} to a
87     * <code>LexicalHandler</code>.</p>
88     *
89     * @param handler A non-null <code>LexicalHandler</code> for
90     * handling lexical parse events.
91     */
92     public void setLexicalHandler(LexicalHandler handler) {
93         this.lexhandler = handler;
94     }
95
96     /**
97     * Get a SAX2 {@link org.xml.sax.ext.LexicalHandler} for the output.
98     *
99     * @return A <code>LexicalHandler</code>, or null.
100    */
101    public LexicalHandler getLexicalHandler() {
102        return lexhandler;
103    }
104
105    /**
106     * Method setSystemId Set the systemID that may be used in association
107     * with the {@link org.xml.sax.ContentHandler}.
108     *
109     * @param systemId The system identifier as a URI string.
110     */
111    public void setSystemId(String systemId) {
112        this.systemId = systemId;
113    }
114
115    /**
116     * Get the system identifier that was set with setSystemId.
117     *
118     * @return The system identifier that was set with setSystemId, or null
119     * if setSystemId was not called.
120     */
121    public String getSystemId() {
122        return systemId;
123    }
124
125    ///////////////////////////////////////////////////
126    // Internal state.
127    ///////////////////////////////////////////////////
128
129    /**
130     * The handler for parse events.
131     */
132    private ContentHandler handler;
133
134    /**
135     * The handler for lexical events.
136     */
137    private LexicalHandler lexhandler;

```

```

138
139 /**
140  * The systemID that may be used in association
141  * with the node.
142  */
143 private String systemId;
144 }

```

## 23.2 SAXSource.java

Secuencia 243: javax/xml/transform/sax/SAXSource.java

```

1  /*
2  * Copyright (c) 2000, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.sax;
27
28 import javax.xml.transform.Source;
29 import javax.xml.transform.stream.StreamSource;
30
31 import org.xml.sax.InputSource;
32 import org.xml.sax.XMLReader;
33
34 /**
35  * <p>Acts as an holder for SAX-style Source.</p>
36  *
37  * <p>Note that XSLT requires namespace support. Attempting to transform an
38  * input source that is not
39  * generated with a namespace-aware parser may result in errors.
40  * Parsers can be made namespace aware by calling the
41  * {@link javax.xml.parsers.SAXParserFactory#setNamespaceAware(boolean awareness)} method.</p>
42  *

```

```

43  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
44  */
45  public class SAXSource implements Source {
46
47      /**
48       * If {@link javax.xml.transform.TransformerFactory#getFeature}
49       * returns true when passed this value as an argument,
50       * the Transformer supports Source input of this type.
51       */
52      public static final String FEATURE =
53          "http://javax.xml.transform.sax.SAXSource/feature";
54
55      /**
56       * <p>Zero-argument default constructor. If this constructor is used, and
57       * no SAX source is set using
58       * {@link #setInputSource(InputSource inputSource)} , then the
59       * <code>Transformer</code> will
60       * create an empty source {@link org.xml.sax.InputSource} using
61       * {@link org.xml.sax.InputSource#InputSource() new InputSource()}.</p>
62       *
63       * @see javax.xml.transform.Transformer#transform(Source xmlSource, Result outputTarget)
64       */
65      public SAXSource() { }
66
67      /**
68       * Create a <code>SAXSource</code>, using an {@link org.xml.sax.XMLReader}
69       * and a SAX InputSource. The {@link javax.xml.transform.Transformer}
70       * or {@link javax.xml.transform.sax.SAXTransformerFactory} will set itself
71       * to be the reader's {@link org.xml.sax.ContentHandler}, and then will call
72       * reader.parse(inputSource).
73       *
74       * @param reader An XMLReader to be used for the parse.
75       * @param inputSource A SAX input source reference that must be non-null
76       * and that will be passed to the reader parse method.
77       */
78      public SAXSource(XMLReader reader, InputSource inputSource) {
79          this.reader = reader;
80          this.inputSource = inputSource;
81      }
82
83      /**
84       * Create a <code>SAXSource</code>, using a SAX <code>InputSource</code>.
85       * The {@link javax.xml.transform.Transformer} or
86       * {@link javax.xml.transform.sax.SAXTransformerFactory} creates a
87       * reader via {@link org.xml.sax.helpers.XMLReaderFactory}
88       * (if setXMLReader is not used), sets itself as
89       * the reader's {@link org.xml.sax.ContentHandler}, and calls
90       * reader.parse(inputSource).
91       *
92       * @param inputSource An input source reference that must be non-null
93       * and that will be passed to the parse method of the reader.
94       */
95      public SAXSource(InputSource inputSource) {

```

```
96         this.inputSource = inputSource;
97     }
98
99     /**
100      * Set the XMLReader to be used for the Source.
101      *
102      * @param reader A valid XMLReader or XMLFilter reference.
103      */
104     public void setXMLReader(XMLReader reader) {
105         this.reader = reader;
106     }
107
108     /**
109      * Get the XMLReader to be used for the Source.
110      *
111      * @return A valid XMLReader or XMLFilter reference, or null.
112      */
113     public XMLReader getXMLReader() {
114         return reader;
115     }
116
117     /**
118      * Set the SAX InputSource to be used for the Source.
119      *
120      * @param inputSource A valid InputSource reference.
121      */
122     public void setInputSource(InputSource inputSource) {
123         this.inputSource = inputSource;
124     }
125
126     /**
127      * Get the SAX InputSource to be used for the Source.
128      *
129      * @return A valid InputSource reference, or null.
130      */
131     public InputSource getInputSource() {
132         return inputSource;
133     }
134
135     /**
136      * Set the system identifier for this Source. If an input source
137      * has already been set, it will set the system ID or that
138      * input source, otherwise it will create a new input source.
139      *
140      * <p>The system identifier is optional if there is a byte stream
141      * or a character stream, but it is still useful to provide one,
142      * since the application can use it to resolve relative URIs
143      * and can include it in error messages and warnings (the parser
144      * will attempt to open a connection to the URI only if
145      * no byte stream or character stream is specified).</p>
146      *
147      * @param systemId The system identifier as a URI string.
148      */
```

```

149     public void setSystemId(String systemId) {
150
151         if (null == inputSource) {
152             inputSource = new InputSource(systemId);
153         } else {
154             inputSource.setSystemId(systemId);
155         }
156     }
157
158     /**
159     * <p>Get the base ID (URI or system ID) from where URIs
160     * will be resolved.</p>
161     *
162     * @return Base URL for the <code>Source</code>, or <code>null</code>.
163     */
164     public String getSystemId() {
165
166         if (inputSource == null) {
167             return null;
168         } else {
169             return inputSource.getSystemId();
170         }
171     }
172
173     /**
174     * The XMLReader to be used for the source tree input. May be null.
175     */
176     private XMLReader reader;
177
178     /**
179     * <p>The SAX InputSource to be used for the source tree input.
180     * Should not be <code>null</code>.</p>
181     */
182     private InputSource inputSource;
183
184     /**
185     * Attempt to obtain a SAX InputSource object from a Source
186     * object.
187     *
188     * @param source Must be a non-null Source reference.
189     *
190     * @return An InputSource, or null if Source can not be converted.
191     */
192     public static InputSource sourceToInputSource(Source source) {
193
194         if (source instanceof SAXSource) {
195             return ((SAXSource) source).getInputSource();
196         } else if (source instanceof StreamSource) {
197             StreamSource ss = (StreamSource) source;
198             InputSource isource = new InputSource(ss.getSystemId());
199
200             isource.setByteStream(ss.getInputStream());
201             isource.setCharacterStream(ss.getReader());

```

```

202         isource.setPublicId(ss.getPublicId());
203
204         return isource;
205     } else {
206         return null;
207     }
208 }
209 }

```

### 23.3 SAXTransformerFactory.java

Secuencia 244: javax/xml/transform/sax/SAXTransformerFactory.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.sax;
27
28 import javax.xml.transform.*;
29
30 import org.xml.sax.XMLFilter;
31
32 /**
33  * This class extends TransformerFactory to provide SAX-specific
34  * factory methods. It provides two types of ContentHandlers,
35  * one for creating Transformers, the other for creating Templates
36  * objects.
37  *
38  * <p>If an application wants to set the ErrorHandler or EntityResolver
39  * for an XMLReader used during a transformation, it should use a URIResolver
40  * to return the SAXSource which provides (with getXMLReader) a reference to
41  * the XMLReader.</p>

```



```

42  */
43  public abstract class SAXTransformerFactory extends TransformerFactory {
44
45      /** If {@link javax.xml.transform.TransformerFactory#getFeature}
46       * returns true when passed this value as an argument,
47       * the TransformerFactory returned from
48       * {@link javax.xml.transform.TransformerFactory#newInstance} may
49       * be safely cast to a SAXTransformerFactory.
50       */
51      public static final String FEATURE =
52          "http://javax.xml.transform.sax.SAXTransformerFactory/feature";
53
54      /** If {@link javax.xml.transform.TransformerFactory#getFeature}
55       * returns true when passed this value as an argument,
56       * the {@link #newXMLFilter(Source src)}
57       * and {@link #newXMLFilter(Templates templates)} methods are supported.
58       */
59      public static final String FEATURE_XMLFILTER =
60          "http://javax.xml.transform.sax.SAXTransformerFactory/feature/xmlfilter";
61
62      /**
63       * The default constructor is protected on purpose.
64       */
65      protected SAXTransformerFactory() {}
66
67      /**
68       * Get a TransformerHandler object that can process SAX
69       * ContentHandler events into a Result, based on the transformation
70       * instructions specified by the argument.
71       *
72       * @param src The Source of the transformation instructions.
73       *
74       * @return TransformerHandler ready to transform SAX events.
75       *
76       * @throws TransformerConfigurationException If for some reason the
77       * TransformerHandler can not be created.
78       */
79      public abstract TransformerHandler newTransformerHandler(Source src)
80          throws TransformerConfigurationException;
81
82      /**
83       * Get a TransformerHandler object that can process SAX
84       * ContentHandler events into a Result, based on the Templates argument.
85       *
86       * @param templates The compiled transformation instructions.
87       *
88       * @return TransformerHandler ready to transform SAX events.
89       *
90       * @throws TransformerConfigurationException If for some reason the
91       * TransformerHandler can not be created.
92       */
93      public abstract TransformerHandler newTransformerHandler(
94          Templates templates) throws TransformerConfigurationException;

```

```
95
96 /**
97  * Get a TransformerHandler object that can process SAX
98  * ContentHandler events into a Result. The transformation
99  * is defined as an identity (or copy) transformation, for example
100  * to copy a series of SAX parse events into a DOM tree.
101  *
102  * @return A non-null reference to a TransformerHandler, that may
103  * be used as a ContentHandler for SAX parse events.
104  *
105  * @throws TransformerConfigurationException If for some reason the
106  * TransformerHandler cannot be created.
107  */
108 public abstract TransformerHandler newTransformerHandler()
109     throws TransformerConfigurationException;
110
111 /**
112  * Get a TemplatesHandler object that can process SAX
113  * ContentHandler events into a Templates object.
114  *
115  * @return A non-null reference to a TransformerHandler, that may
116  * be used as a ContentHandler for SAX parse events.
117  *
118  * @throws TransformerConfigurationException If for some reason the
119  * TemplatesHandler cannot be created.
120  */
121 public abstract TemplatesHandler newTemplatesHandler()
122     throws TransformerConfigurationException;
123
124 /**
125  * Create an XMLFilter that uses the given Source as the
126  * transformation instructions.
127  *
128  * @param src The Source of the transformation instructions.
129  *
130  * @return An XMLFilter object, or null if this feature is not supported.
131  *
132  * @throws TransformerConfigurationException If for some reason the
133  * TemplatesHandler cannot be created.
134  */
135 public abstract XMLFilter newXMLFilter(Source src)
136     throws TransformerConfigurationException;
137
138 /**
139  * Create an XMLFilter, based on the Templates argument..
140  *
141  * @param templates The compiled transformation instructions.
142  *
143  * @return An XMLFilter object, or null if this feature is not supported.
144  *
145  * @throws TransformerConfigurationException If for some reason the
146  * TemplatesHandler cannot be created.
147  */
```

```

148     public abstract XMLFilter newXMLFilter(Templates templates)
149         throws TransformerConfigurationException;
150 }

```

## 23.4 TemplatesHandler.java

Secuencia 245: javax/xml/transform/sax/TemplatesHandler.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.sax;
27
28 import javax.xml.transform.*;
29
30 import org.xml.sax.ContentHandler;
31
32 /**
33  * A SAX ContentHandler that may be used to process SAX
34  * parse events (parsing transformation instructions) into a Templates object.
35  *
36  * <p>Note that TemplatesHandler does not need to implement LexicalHandler.</p>
37  */
38 public interface TemplatesHandler extends ContentHandler {
39
40     /**
41      * When a TemplatesHandler object is used as a ContentHandler
42      * for the parsing of transformation instructions, it creates a Templates object,
43      * which the caller can get once the SAX events have been completed.
44      *
45      * @return The Templates object that was created during
46      * the SAX event process, or null if no Templates object has

```

```

47     * been created.
48     *
49     */
50     public Templates getTemplates();
51
52     /**
53     * Set the base ID (URI or system ID) for the Templates object
54     * created by this builder. This must be set in order to
55     * resolve relative URIs in the stylesheet. This must be
56     * called before the startDocument event.
57     *
58     * @param systemID Base URI for this stylesheet.
59     */
60     public void setSystemId(String systemID);
61
62     /**
63     * Get the base ID (URI or system ID) from where relative
64     * URLs will be resolved.
65     * @return The systemID that was set with {@link #setSystemId}.
66     */
67     public String getSystemId();
68 }

```

## 23.5 TransformerHandler.java

Secuencia 246: javax.xml.transform.sax/TransformerHandler.java

```

1  /*
2  * Copyright (c) 2000, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.sax;
27

```

```
28 import javax.xml.transform.Result;
29 import javax.xml.transform.Transformer;
30
31 import org.xml.sax.ContentHandler;
32 import org.xml.sax.DTDHandler;
33 import org.xml.sax.ext.LexicalHandler;
34
35 /**
36  * A TransformerHandler
37  * listens for SAX ContentHandler parse events and transforms
38  * them to a Result.
39  */
40 public interface TransformerHandler
41     extends ContentHandler, LexicalHandler, DTDHandler {
42
43     /**
44      * <p>Set the <code>Result</code> associated with this
45      * <code>TransformerHandler</code> to be used for the transformation.</p>
46      *
47      * @param result A <code>Result</code> instance, should not be
48      * <code>null</code>.
49      *
50      * @throws IllegalArgumentException if result is invalid for some reason.
51      */
52     public void setResult(Result result) throws IllegalArgumentException;
53
54     /**
55      * Set the base ID (URI or system ID) from where relative
56      * URLs will be resolved.
57      * @param systemID Base URI for the source tree.
58      */
59     public void setSystemId(String systemID);
60
61     /**
62      * Get the base ID (URI or system ID) from where relative
63      * URLs will be resolved.
64      * @return The systemID that was set with {@link #setSystemId}.
65      */
66     public String getSystemId();
67
68     /**
69      * <p>Get the <code>Transformer</code> associated with this handler, which
70      * is needed in order to set parameters and output properties.</p>
71      *
72      * @return <code>Transformer</code> associated with this
73      * <code>TransformerHandler</code>.
74      */
75     public Transformer getTransformer();
76 }
```

## 24 Xml Transform Stax (javax.xml.transform.stax package)

### 24.1 StAXResult.java

Secuencia 247: javax/xml/transform/stax/StAXResult.java

```
1  /*
2  * Copyright (c) 2005, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.stax;
27
28 import javax.xml.stream.XMLEventWriter;
29 import javax.xml.stream.XMLStreamWriter;
30 import javax.xml.transform.Result;
31
32 /**
33  * <p>Acts as a holder for an XML {@link Result} in the
34  * form of a StAX writer, i.e.
35  * {@link XMLStreamWriter} or {@link XMLEventWriter}.
36  * <code>StAXResult</code> can be used in all cases that accept
37  * a <code>Result</code>, e.g. {@link javax.xml.transform.Transformer},
38  * {@link javax.xml.validation.Validator} which accept
39  * <code>Result</code> as input.
40  *
41  * @author <a href="mailto:Neeraj.Bajaj@Sun.com">Neeraj Bajaj</a>
42  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
43  *
44  * @see <a href="http://jcp.org/en/jsr/detail?id=173">
45  *   JSR 173: Streaming API for XML</a>
46  * @see XMLStreamWriter
47  * @see XMLEventWriter
```

```

48  *
49  * @since 1.6
50  */
51  public class StAXResult implements Result {
52      /** If {@link javax.xml.transform.TransformerFactory#getFeature(String name)}
53       * returns true when passed this value as an argument,
54       * the Transformer supports Result output of this type.
55       */
56      public static final String FEATURE =
57          "http://javax.xml.transform.stax.StAXResult/feature";
58
59      /**
60       * <p><code>XMLEventWriter</code> to be used for
61       * <code>Result</code> output.</p>
62       */
63      private XMLEventWriter xmlEventWriter = null;
64
65      /**
66       * <p><code>XMLStreamWriter</code> to be used for
67       * <code>Result</code> output.</p>
68       */
69      private XMLStreamWriter xmlStreamWriter = null;
70
71      /** <p>System identifier for this <code>StAXResult</code>.<p> */
72      private String systemId = null;
73
74      /**
75       * <p>Creates a new instance of a <code>StAXResult</code>
76       * by supplying an {@link XMLEventWriter}.</p>
77       *
78       * <p><code>XMLEventWriter</code> must be a
79       * non-<code>>null</code> reference.</p>
80       *
81       * @param xmlEventWriter <code>XMLEventWriter</code> used to create
82       * this <code>StAXResult</code>.
83       *
84       * @throws IllegalArgumentException If <code>xmlEventWriter</code> ==
85       * <code>>null</code>.
86       */
87      public StAXResult(final XMLEventWriter xmlEventWriter) {
88
89          if (xmlEventWriter == null) {
90              throw new IllegalArgumentException(
91                  "StAXResult(XMLEventWriter) with XMLEventWriter == null");
92          }
93
94          this.xmlEventWriter = xmlEventWriter;
95      }
96
97      /**
98       * <p>Creates a new instance of a <code>StAXResult</code>
99       * by supplying an {@link XMLStreamWriter}.</p>
100      *

```

```

101 * <p><code>XMLStreamWriter</code> must be a
102 * non-<code>null</code> reference.</p>
103 *
104 * @param xmlStreamWriter <code>XMLStreamWriter</code> used to create
105 * this <code>StAXResult</code>.
106 *
107 * @throws IllegalArgumentException If <code>xmlStreamWriter</code> ==
108 * <code>null</code>.
109 */
110 public StAXResult(final XMLStreamWriter xmlStreamWriter) {
111
112     if (xmlStreamWriter == null) {
113         throw new IllegalArgumentException(
114             "StAXResult(XMLStreamWriter) with XMLStreamWriter == null");
115     }
116
117     this.xmlStreamWriter = xmlStreamWriter;
118 }
119
120 /**
121 * <p>Get the <code>XMLEventWriter</code> used by this
122 * <code>StAXResult</code>.</p>
123 *
124 * <p><code>XMLEventWriter</code> will be <code>null</code>
125 * if this <code>StAXResult</code> was created with a
126 * <code>XMLStreamWriter</code>.</p>
127 *
128 * @return <code>XMLEventWriter</code> used by this
129 * <code>StAXResult</code>.
130 */
131 public XMLEventWriter getXMLEventWriter() {
132
133     return xmlEventWriter;
134 }
135
136 /**
137 * <p>Get the <code>XMLStreamWriter</code> used by this
138 * <code>StAXResult</code>.</p>
139 *
140 * <p><code>XMLStreamWriter</code> will be <code>null</code>
141 * if this <code>StAXResult</code> was created with a
142 * <code>XMLEventWriter</code>.</p>
143 *
144 * @return <code>XMLStreamWriter</code> used by this
145 * <code>StAXResult</code>.
146 */
147 public XMLStreamWriter getXMLStreamWriter() {
148
149     return xmlStreamWriter;
150 }
151
152 /**
153 * <p>In the context of a <code>StAXResult</code>, it is not appropriate

```



```

154     * to explicitly set the system identifier.
155     * The <code>XMLStreamWriter</code> or <code>XMLStreamWriter</code>
156     * used to construct this <code>StAXResult</code> determines the
157     * system identifier of the XML result.</p>
158     *
159     * <p>An {@link UnsupportedOperationException} is <strong>always</strong>
160     * thrown by this method.</p>
161     *
162     * @param systemId Ignored.
163     *
164     * @throws UnsupportedOperationException Is <strong>always</strong>
165     *   thrown by this method.
166     */
167     public void setSystemId(final String systemId) {
168
169         throw new UnsupportedOperationException(
170             "StAXResult#setSystemId(systemId) cannot set the "
171             + "system identifier for a StAXResult");
172     }
173
174     /**
175     * <p>The returned system identifier is always <code>null</code>.</p>
176     *
177     * @return The returned system identifier is always <code>null</code>.
178     */
179     public String getSystemId() {
180
181         return null;
182     }
183 }

```

## 24.2 StAXSource.java

Secuencia 248: javax/xml/transform/stax/StAXSource.java

```

1  /*
2  * Copyright (c) 2005, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```

```

20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.transform.stax;
27
28 import javax.xml.stream.XMLEventReader;
29 import javax.xml.stream.XMLStreamConstants;
30 import javax.xml.stream.XMLStreamException;
31 import javax.xml.stream.XMLStreamReader;
32 import javax.xml.stream.events.XMLEvent;
33 import javax.xml.transform.Source;
34
35 /**
36  * <p>Acts as a holder for an XML {@link Source} in the
37  * form of a StAX reader, i.e.
38  * {@link XMLStreamReader} or {@link XMLEventReader}.
39  * <code>StAXSource</code> can be used in all cases that accept
40  * a <code>Source</code>, e.g. {@link javax.xml.transform.Transformer},
41  * {@link javax.xml.validation.Validator} which accept
42  * <code>Source</code> as input.
43  *
44  * <p><code>StAXSource</code>s are consumed during processing
45  * and are not reusable.</p>
46  *
47  * @author <a href="mailto:Neeraj.Bajaj@Sun.com">Neeraj Bajaj</a>
48  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
49  *
50  * @see <a href="http://jcp.org/en/jsr/detail?id=173">
51  *   JSR 173: Streaming API for XML</a>
52  * @see XMLStreamReader
53  * @see XMLEventReader
54  *
55  * @since 1.6
56  */
57 public class StAXSource implements Source {
58
59     /** If {@link javax.xml.transform.TransformerFactory#getFeature(String name)}
60      * returns true when passed this value as an argument,
61      * the Transformer supports Source input of this type.
62      */
63     public static final String FEATURE =
64         "http://javax.xml.transform.stax.StAXSource/feature";
65
66     /** <p><code>XMLEventReader</code> to be used for source input.</p> */
67     private XMLEventReader xmlEventReader = null;
68
69     /** <p><code>XMLStreamReader</code> to be used for source input.</p> */
70     private XMLStreamReader xmlStreamReader = null;
71
72     /** <p>System identifier of source input.</p> */

```

```

73     private String systemId = null;
74
75     /**
76      * <p>Creates a new instance of a <code>StAXSource</code>
77      * by supplying an {@link XMLEventReader}.</p>
78      *
79      * <p><code>XMLEventReader</code> must be a
80      * non-<code>null</code> reference.</p>
81      *
82      * <p><code>XMLEventReader</code> must be in
83      * {@link XMLStreamConstants#START_DOCUMENT} or
84      * {@link XMLStreamConstants#START_ELEMENT} state.</p>
85      *
86      * @param xmlEventReader <code>XMLEventReader</code> used to create
87      * this <code>StAXSource</code>.
88      *
89      * @throws XMLStreamException If <code>xmlEventReader</code> access
90      * throws an <code>Exception</code>.
91      * @throws IllegalArgumentException If <code>xmlEventReader</code> ==
92      * <code>null</code>.
93      * @throws IllegalStateException If <code>xmlEventReader</code>
94      * is not in <code>XMLStreamConstants.START_DOCUMENT</code> or
95      * <code>XMLStreamConstants.START_ELEMENT</code> state.
96      */
97     public StAXSource(final XMLEventReader xmlEventReader)
98         throws XMLStreamException {
99
100         if (xmlEventReader == null) {
101             throw new IllegalArgumentException(
102                 "StAXSource(XMLEventReader) with XMLEventReader == null");
103         }
104
105         // TODO: This is ugly ...
106         // there is no way to know the current position(event) of
107         // XMLEventReader. peek() is the only way to know the next event.
108         // The next event on the input stream should be
109         // XMLStreamConstants.START_DOCUMENT or
110         // XMLStreamConstants.START_ELEMENT.
111         XMLEvent event = xmlEventReader.peek();
112         int eventType = event.getEventType();
113         if (eventType != XMLStreamConstants.START_DOCUMENT
114             && eventType != XMLStreamConstants.START_ELEMENT) {
115             throw new IllegalStateException(
116                 "StAXSource(XMLEventReader) with XMLEventReader "
117                 + "not in XMLStreamConstants.START_DOCUMENT or "
118                 + "XMLStreamConstants.START_ELEMENT state");
119         }
120
121         this.xmlEventReader = xmlEventReader;
122         systemId = event.getLocation().getSystemId();
123     }
124
125     /**

```

```

126 * <p>Creates a new instance of a <code>StAXSource</code>
127 * by supplying an {@link XMLStreamReader}.</p>
128 *
129 * <p><code>XMLStreamReader</code> must be a
130 * non-<code>null</code> reference.</p>
131 *
132 * <p><code>XMLStreamReader</code> must be in
133 * {@link XMLStreamConstants#START_DOCUMENT} or
134 * {@link XMLStreamConstants#START_ELEMENT} state.</p>
135 *
136 * @param xmlStreamReader <code>XMLStreamReader</code> used to create
137 * this <code>StAXSource</code>.
138 *
139 * @throws IllegalArgumentException If <code>xmlStreamReader</code> ==
140 * <code>null</code>.
141 * @throws IllegalStateException If <code>xmlStreamReader</code>
142 * is not in <code>XMLStreamConstants.START_DOCUMENT</code> or
143 * <code>XMLStreamConstants.START_ELEMENT</code> state.
144 */
145 public StAXSource(final XMLStreamReader xmlStreamReader) {
146
147     if (xmlStreamReader == null) {
148         throw new IllegalArgumentException(
149             "StAXSource(XMLStreamReader) with XMLStreamReader == null");
150     }
151
152     int eventType = xmlStreamReader.getEventType();
153     if (eventType != XMLStreamConstants.START_DOCUMENT
154         && eventType != XMLStreamConstants.START_ELEMENT) {
155         throw new IllegalStateException(
156             "StAXSource(XMLStreamReader) with XMLStreamReader"
157             + "not in XMLStreamConstants.START_DOCUMENT or "
158             + "XMLStreamConstants.START_ELEMENT state");
159     }
160
161     this.xmlStreamReader = xmlStreamReader;
162     systemId = xmlStreamReader.getLocation().getSystemId();
163 }
164
165 /**
166 * <p>Get the <code>XMLEventReader</code> used by this
167 * <code>StAXSource</code>.</p>
168 *
169 * <p><code>XMLEventReader</code> will be <code>null</code>.
170 * if this <code>StAXSource</code> was created with a
171 * <code>XMLStreamReader</code>.</p>
172 *
173 * @return <code>XMLEventReader</code> used by this
174 * <code>StAXSource</code>.
175 */
176 public XMLEventReader getXMLEventReader() {
177
178     return xmlEventReader;

```

```

179     }
180
181     /**
182     * <p>Get the <code>XMLStreamReader</code> used by this
183     * <code>StAXSource</code>.</p>
184     *
185     * <p><code>XMLStreamReader</code> will be <code>null</code>
186     * if this <code>StAXSource</code> was created with a
187     * <code>XMLEventReader</code>.</p>
188     *
189     * @return <code>XMLStreamReader</code> used by this
190     *         <code>StAXSource</code>.
191     */
192     public XMLStreamReader getXMLStreamReader() {
193
194         return xmlStreamReader;
195     }
196
197     /**
198     * <p>In the context of a <code>StAXSource</code>, it is not appropriate
199     * to explicitly set the system identifier.
200     * The <code>XMLStreamReader</code> or <code>XMLEventReader</code>
201     * used to construct this <code>StAXSource</code> determines the
202     * system identifier of the XML source.</p>
203     *
204     * <p>An {@link UnsupportedOperationException} is <strong>always</strong>
205     * thrown by this method.</p>
206     *
207     * @param systemId Ignored.
208     *
209     * @throws UnsupportedOperationException Is <strong>always</strong>
210     *         thrown by this method.
211     */
212     public void setSystemId(final String systemId) {
213
214         throw new UnsupportedOperationException(
215             "StAXSource#setSystemId(systemId) cannot set the "
216             + "system identifier for a StAXSource");
217     }
218
219     /**
220     * <p>Get the system identifier used by this
221     * <code>StAXSource</code>.</p>
222     *
223     * <p>The <code>XMLStreamReader</code> or <code>XMLEventReader</code>
224     * used to construct this <code>StAXSource</code> is queried to determine
225     * the system identifier of the XML source.</p>
226     *
227     * <p>The system identifier may be <code>null</code> or
228     * an empty <code>""</code> <code>String</code>.</p>
229     *
230     * @return System identifier used by this <code>StAXSource</code>.
231     */

```

```
232     public String getSystemId() {  
233  
234         return systemId;  
235     }  
236 }
```

## 25 Xml Transform Stream (javax.xml.transform.stream package)

### 25.1 StreamResult.java

Secuencia 249: javax/xml/transform/stream/StreamResult.java

```
1  /*  
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.  
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.  
4  *  
5  *  
6  *  
7  *  
8  *  
9  *  
10 *  
11 *  
12 *  
13 *  
14 *  
15 *  
16 *  
17 *  
18 *  
19 *  
20 *  
21 *  
22 *  
23 *  
24 */  
25  
26 package javax.xml.transform.stream;  
27  
28 import javax.xml.transform.Result;  
29  
30 import java.io.File;  
31 import java.io.OutputStream;  
32 import java.io.Writer;  
33 import java.net.MalformedURLException;  
34  
35 /**  
36  * <p>Acts as an holder for a transformation result,  
37  * which may be XML, plain Text, HTML, or some other form of markup.</p>  
38  *  
39  * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>  
40  */  
41 public class StreamResult implements Result {  
42
```

```
43  /** If {@link javax.xml.transform.TransformerFactory#getFeature}
44  * returns true when passed this value as an argument,
45  * the Transformer supports Result output of this type.
46  */
47  public static final String FEATURE =
48      "http://javax.xml.transform.stream.StreamResult/feature";
49
50  /**
51   * Zero-argument default constructor.
52   */
53  public StreamResult() {
54  }
55
56  /**
57   * Construct a StreamResult from a byte stream. Normally,
58   * a stream should be used rather than a reader, so that
59   * the transformer may use instructions contained in the
60   * transformation instructions to control the encoding.
61   *
62   * @param outputStream A valid OutputStream reference.
63   */
64  public StreamResult(OutputStream outputStream) {
65      setOutputStream(outputStream);
66  }
67
68  /**
69   * Construct a StreamResult from a character stream. Normally,
70   * a stream should be used rather than a reader, so that
71   * the transformer may use instructions contained in the
72   * transformation instructions to control the encoding. However,
73   * there are times when it is useful to write to a character
74   * stream, such as when using a StringWriter.
75   *
76   * @param writer A valid Writer reference.
77   */
78  public StreamResult(Writer writer) {
79      setWriter(writer);
80  }
81
82  /**
83   * Construct a StreamResult from a URL.
84   *
85   * @param systemId Must be a String that conforms to the URI syntax.
86   */
87  public StreamResult(String systemId) {
88      this.systemId = systemId;
89  }
90
91  /**
92   * Construct a StreamResult from a File.
93   *
94   * @param f Must a non-null File reference.
95   */
```

```
96     public StreamResult(File f) {
97         //convert file to appropriate URI, f.toURI().toASCIIString()
98         //converts the URI to string as per rule specified in
99         //RFC 2396,
100         setSystemId(f.toURI().toASCIIString());
101     }
102
103     /**
104      * Set the ByteStream that is to be written to. Normally,
105      * a stream should be used rather than a reader, so that
106      * the transformer may use instructions contained in the
107      * transformation instructions to control the encoding.
108      *
109      * @param outputStream A valid OutputStream reference.
110      */
111     public void setOutputStream(OutputStream outputStream) {
112         this.outputStream = outputStream;
113     }
114
115     /**
116      * Get the byte stream that was set with setOutputStream.
117      *
118      * @return The byte stream that was set with setOutputStream, or null
119      *         if setOutputStream or the ByteStream constructor was not called.
120      */
121     public OutputStream getOutputStream() {
122         return outputStream;
123     }
124
125     /**
126      * Set the writer that is to receive the result. Normally,
127      * a stream should be used rather than a writer, so that
128      * the transformer may use instructions contained in the
129      * transformation instructions to control the encoding. However,
130      * there are times when it is useful to write to a writer,
131      * such as when using a StringWriter.
132      *
133      * @param writer A valid Writer reference.
134      */
135     public void setWriter(Writer writer) {
136         this.writer = writer;
137     }
138
139     /**
140      * Get the character stream that was set with setWriter.
141      *
142      * @return The character stream that was set with setWriter, or null
143      *         if setWriter or the Writer constructor was not called.
144      */
145     public Writer getWriter() {
146         return writer;
147     }
148
```



```

149  /**
150   * Set the systemID that may be used in association
151   * with the byte or character stream, or, if neither is set, use
152   * this value as a writeable URI (probably a file name).
153   *
154   * @param systemId The system identifier as a URI string.
155   */
156  public void setSystemId(String systemId) {
157      this.systemId = systemId;
158  }
159
160  /**
161   * <p>Set the system ID from a <code>File</code> reference.</p>
162   *
163   *
164   * @param f Must a non-null File reference.
165   */
166  public void setSystemId(File f) {
167      //convert file to appropriate URI, f.toURI().toASCIIString()
168      //converts the URI to string as per rule specified in
169      //RFC 2396,
170      this.systemId = f.toURI().toASCIIString();
171  }
172
173  /**
174   * Get the system identifier that was set with setSystemId.
175   *
176   * @return The system identifier that was set with setSystemId, or null
177   * if setSystemId was not called.
178   */
179  public String getSystemId() {
180      return systemId;
181  }
182
183  //////////////////////////////////////
184  // Internal state.
185  //////////////////////////////////////
186
187  /**
188   * The systemID that may be used in association
189   * with the byte or character stream, or, if neither is set, use
190   * this value as a writeable URI (probably a file name).
191   */
192  private String systemId;
193
194  /**
195   * The byte stream that is to be written to.
196   */
197  private OutputStream outputStream;
198
199  /**
200   * The character stream that is to be written to.
201   */

```

```

202     private Writer writer;
203 }

```

## 25.2 StreamSource.java

Secuencia 250: javax/xml/transform/stream/StreamSource.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.transform.stream;
27
28 import java.io.File;
29 import java.io.InputStream;
30 import java.io.Reader;
31
32 import javax.xml.transform.Source;
33
34 /**
35  * <p>Acts as an holder for a transformation Source in the form
36  * of a stream of XML markup.</p>
37  *
38  * <p><em>Note:</em> Due to their internal use of either a {@link Reader} or {@link InputStream}
39  * instance,
40  * <code>StreamSource</code> instances may only be used once.</p>
41  * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>
42  */
43 public class StreamSource implements Source {
44
45     /** If {@link javax.xml.transform.TransformerFactory#getFeature}
46     * returns true when passed this value as an argument,

```

```

47     * the Transformer supports Source input of this type.
48     */
49     public static final String FEATURE =
50         "http://javax.xml.transform.stream.StreamSource/feature";
51
52     /**
53     * <p>Zero-argument default constructor. If this constructor is used, and
54     * no Stream source is set using
55     * {@link #setInputStream(java.io.InputStream inputStream)} or
56     * {@link #setReader(java.io.Reader reader)}, then the
57     * <code>Transformer</code> will
58     * create an empty source {@link java.io.InputStream} using
59     * {@link java.io.InputStream#InputStream() new InputStream()}.</p>
60     *
61     * @see javax.xml.transform.Transformer#transform(Source xmlSource, Result outputTarget)
62     */
63     public StreamSource() { }
64
65     /**
66     * Construct a StreamSource from a byte stream. Normally,
67     * a stream should be used rather than a reader, so
68     * the XML parser can resolve character encoding specified
69     * by the XML declaration.
70     *
71     * <p>If this constructor is used to process a stylesheet, normally
72     * setSystemId should also be called, so that relative URI references
73     * can be resolved.</p>
74     *
75     * @param inputStream A valid InputStream reference to an XML stream.
76     */
77     public StreamSource(InputStream inputStream) {
78         setInputStream(inputStream);
79     }
80
81     /**
82     * Construct a StreamSource from a byte stream. Normally,
83     * a stream should be used rather than a reader, so that
84     * the XML parser can resolve character encoding specified
85     * by the XML declaration.
86     *
87     * <p>This constructor allows the systemID to be set in addition
88     * to the input stream, which allows relative URIs
89     * to be processed.</p>
90     *
91     * @param inputStream A valid InputStream reference to an XML stream.
92     * @param systemId Must be a String that conforms to the URI syntax.
93     */
94     public StreamSource(InputStream inputStream, String systemId) {
95         setInputStream(inputStream);
96         setSystemId(systemId);
97     }
98
99     /**

```

```
100     * Construct a StreamSource from a character reader. Normally,
101     * a stream should be used rather than a reader, so that
102     * the XML parser can resolve character encoding specified
103     * by the XML declaration. However, in many cases the encoding
104     * of the input stream is already resolved, as in the case of
105     * reading XML from a StringReader.
106     *
107     * @param reader A valid Reader reference to an XML character stream.
108     */
109     public StreamSource(Reader reader) {
110         setReader(reader);
111     }
112
113     /**
114     * Construct a StreamSource from a character reader. Normally,
115     * a stream should be used rather than a reader, so that
116     * the XML parser may resolve character encoding specified
117     * by the XML declaration. However, in many cases the encoding
118     * of the input stream is already resolved, as in the case of
119     * reading XML from a StringReader.
120     *
121     * @param reader A valid Reader reference to an XML character stream.
122     * @param systemId Must be a String that conforms to the URI syntax.
123     */
124     public StreamSource(Reader reader, String systemId) {
125         setReader(reader);
126         setSystemId(systemId);
127     }
128
129     /**
130     * Construct a StreamSource from a URL.
131     *
132     * @param systemId Must be a String that conforms to the URI syntax.
133     */
134     public StreamSource(String systemId) {
135         this.systemId = systemId;
136     }
137
138     /**
139     * Construct a StreamSource from a File.
140     *
141     * @param f Must a non-null File reference.
142     */
143     public StreamSource(File f) {
144         //convert file to appropriate URI, f.toURI().toASCIIString()
145         //converts the URI to string as per rule specified in
146         //RFC 2396,
147         setSystemId(f.toURI().toASCIIString());
148     }
149
150     /**
151     * Set the byte stream to be used as input. Normally,
152     * a stream should be used rather than a reader, so that
```

```
153     * the XML parser can resolve character encoding specified
154     * by the XML declaration.
155     *
156     * <p>If this Source object is used to process a stylesheet, normally
157     * setSystemId should also be called, so that relative URL references
158     * can be resolved.</p>
159     *
160     * @param inputStream A valid InputStream reference to an XML stream.
161     */
162     public void setInputStream(InputStream inputStream) {
163         this.inputStream = inputStream;
164     }
165
166     /**
167     * Get the byte stream that was set with setByteStream.
168     *
169     * @return The byte stream that was set with setByteStream, or null
170     * if setByteStream or the ByteStream constructor was not called.
171     */
172     public InputStream getInputStream() {
173         return inputStream;
174     }
175
176     /**
177     * Set the input to be a character reader. Normally,
178     * a stream should be used rather than a reader, so that
179     * the XML parser can resolve character encoding specified
180     * by the XML declaration. However, in many cases the encoding
181     * of the input stream is already resolved, as in the case of
182     * reading XML from a StringReader.
183     *
184     * @param reader A valid Reader reference to an XML CharacterStream.
185     */
186     public void setReader(Reader reader) {
187         this.reader = reader;
188     }
189
190     /**
191     * Get the character stream that was set with setReader.
192     *
193     * @return The character stream that was set with setReader, or null
194     * if setReader or the Reader constructor was not called.
195     */
196     public Reader getReader() {
197         return reader;
198     }
199
200     /**
201     * Set the public identifier for this Source.
202     *
203     * <p>The public identifier is always optional: if the application
204     * writer includes one, it will be provided as part of the
205     * location information.</p>
```

```
206     *
207     * @param publicId The public identifier as a string.
208     */
209     public void setPublicId(String publicId) {
210         this.publicId = publicId;
211     }
212
213     /**
214     * Get the public identifier that was set with setPublicId.
215     *
216     * @return The public identifier that was set with setPublicId, or null
217     *         if setPublicId was not called.
218     */
219     public String getPublicId() {
220         return publicId;
221     }
222
223     /**
224     * Set the system identifier for this Source.
225     *
226     * <p>The system identifier is optional if there is a byte stream
227     * or a character stream, but it is still useful to provide one,
228     * since the application can use it to resolve relative URIs
229     * and can include it in error messages and warnings (the parser
230     * will attempt to open a connection to the URI only if
231     * there is no byte stream or character stream specified).</p>
232     *
233     * @param systemId The system identifier as a URL string.
234     */
235     public void setSystemId(String systemId) {
236         this.systemId = systemId;
237     }
238
239     /**
240     * Get the system identifier that was set with setSystemId.
241     *
242     * @return The system identifier that was set with setSystemId, or null
243     *         if setSystemId was not called.
244     */
245     public String getSystemId() {
246         return systemId;
247     }
248
249     /**
250     * Set the system ID from a File reference.
251     *
252     * @param f Must a non-null File reference.
253     */
254     public void setSystemId(File f) {
255         //convert file to appropriate URI, f.toURI().toASCIIString()
256         //converts the URI to string as per rule specified in
257         //RFC 2396,
258         this.systemId = f.toURI().toASCIIString();
```

```

259     }
260
261     //////////////////////////////////////////////////
262     // Internal state.
263     //////////////////////////////////////////////////
264
265     /**
266      * The public identifier for this input source, or null.
267      */
268     private String publicId;
269
270     /**
271      * The system identifier as a URL string, or null.
272      */
273     private String systemId;
274
275     /**
276      * The byte stream for this Source, or null.
277      */
278     private InputStream inputStream;
279
280     /**
281      * The character stream for this Source, or null.
282      */
283     private Reader reader;
284 }

```

## 26 Xml Validation (javax.xml.validation package)

### 26.1 Schema.java

Secuencia 251: javax/xml/validation/Schema.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```

22  *
23  *
24  */
25
26 package javax.xml.validation;
27
28 /**
29  * Immutable in-memory representation of grammar.
30  *
31  * <p>
32  * This object represents a set of constraints that can be checked/
33  * enforced against an XML document.
34  *
35  * <p>
36  * A {@link Schema} object is thread safe and applications are
37  * encouraged to share it across many parsers in many threads.
38  *
39  * <p>
40  * A {@link Schema} object is immutable in the sense that it shouldn't
41  * change the set of constraints once it is created. In other words,
42  * if an application validates the same document twice against the same
43  * {@link Schema}, it must always produce the same result.
44  *
45  * <p>
46  * A {@link Schema} object is usually created from {@link SchemaFactory}.
47  *
48  * <p>
49  * Two kinds of validators can be created from a {@link Schema} object.
50  * One is {@link Validator}, which provides highly-level validation
51  * operations that cover typical use cases. The other is
52  * {@link ValidatorHandler}, which works on top of SAX for better
53  * modularity.
54  *
55  * <p>
56  * This specification does not refine
57  * the {@link java.lang.Object#equals(java.lang.Object)} method.
58  * In other words, if you parse the same schema twice, you may
59  * still get !schemaA.equals(schemaB).
60  *
61  * @author <a href="mailto:Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
62  * @see <a href="http://www.w3.org/TR/xmlschema-1/">XML Schema Part 1: Structures</a>
63  * @see <a href="http://www.w3.org/TR/xml11/">Extensible Markup Language (XML) 1.1</a>
64  * @see <a href="http://www.w3.org/TR/REC-xml">Extensible Markup Language (XML) 1.0 (Second Edition)
65  * </a>
66  * @since 1.5
67  */
68
69 public abstract class Schema {
70
71     /**
72      * Constructor for the derived class.
73      *
74      * <p>
75      * The constructor does nothing.

```



```

74     */
75     protected Schema() {
76     }
77
78     /**
79     * Creates a new {@link Validator} for this {@link Schema}.
80     *
81     * <p>A validator enforces/checks the set of constraints this object
82     * represents.</p>
83     *
84     * <p>Implementors should assure that the properties set on the
85     * {@link SchemaFactory} that created this {@link Schema} are also
86     * set on the {@link Validator} constructed.</p>
87     *
88     * @return
89     *     Always return a non-null valid object.
90     */
91     public abstract Validator newValidator();
92
93     /**
94     * Creates a new {@link ValidatorHandler} for this {@link Schema}.
95     *
96     * <p>Implementors should assure that the properties set on the
97     * {@link SchemaFactory} that created this {@link Schema} are also
98     * set on the {@link ValidatorHandler} constructed.</p>
99     *
100    * @return
101    *     Always return a non-null valid object.
102    */
103    public abstract ValidatorHandler newValidatorHandler();
104 }

```

## 26.2 SchemaFactory.java

Secuencia 252: javax/xml/validation/SchemaFactory.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *

```

```

19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.validation;
27
28 import java.io.File;
29 import java.net.URL;
30 import javax.xml.transform.Source;
31 import javax.xml.transform.stream.StreamSource;
32 import org.w3c.dom.ls.LSResourceResolver;
33 import org.xml.sax.ErrorHandler;
34 import org.xml.sax.SAXException;
35 import org.xml.sax.SAXNotRecognizedException;
36 import org.xml.sax.SAXNotSupportedException;
37 import org.xml.sax.SAXParseException;
38
39 /**
40  * Factory that creates {@link Schema} objects&#x2E; Entry-point to
41  * the validation API.
42  *
43  * <p>
44  * {@link SchemaFactory} is a schema compiler. It reads external
45  * representations of schemas and prepares them for validation.
46  *
47  * <p>
48  * The {@link SchemaFactory} class is not thread-safe. In other words,
49  * it is the application's responsibility to ensure that at most
50  * one thread is using a {@link SchemaFactory} object at any
51  * given moment. Implementations are encouraged to mark methods
52  * as <code>synchronized</code> to protect themselves from broken clients.
53  *
54  * <p>
55  * {@link SchemaFactory} is not re-entrant. While one of the
56  * <code>newSchema</code> methods is being invoked, applications
57  * may not attempt to recursively invoke the <code>newSchema</code> method,
58  * even from the same thread.
59  *
60  * <h2><a name="schemaLanguage"></a>Schema Language</h2>
61  * <p>
62  * This spec uses a namespace URI to designate a schema language.
63  * The following table shows the values defined by this specification.
64  * <p>
65  * To be compliant with the spec, the implementation
66  * is only required to support W3C XML Schema 1.0. However,
67  * if it chooses to support other schema languages listed here,
68  * it must conform to the relevant behaviors described in this spec.
69  *
70  * <p>
71  * Schema languages not listed here are expected to

```

```

72  * introduce their own URIs to represent themselves.
73  * The {@link SchemaFactory} class is capable of locating other
74  * implementations for other schema languages at run-time.
75  *
76  * <p>
77  * Note that because the XML DTD is strongly tied to the parsing process
78  * and has a significant effect on the parsing process, it is impossible
79  * to define the DTD validation as a process independent from parsing.
80  * For this reason, this specification does not define the semantics for
81  * the XML DTD. This doesn't prohibit implementors from implementing it
82  * in a way they see fit, but <em>users are warned that any DTD
83  * validation implemented on this interface necessarily deviate from
84  * the XML DTD semantics as defined in the XML 1.0</em>.
85  *
86  * <table border="1" cellpadding="2">
87  *   <thead>
88  *     <tr>
89  *       <th>value</th>
90  *       <th>language</th>
91  *     </tr>
92  *   </thead>
93  *   <tbody>
94  *     <tr>
95  *       <td>{@link javax.xml.XMLConstants#W3C_XML_SCHEMA_NS_URI} ("<code>http://www.w3.org/2001/
XMLSchema</code>")</td>
96  *       <td><a href="http://www.w3.org/TR/xmlschema-1">W3C XML Schema 1.0</a></td>
97  *     </tr>
98  *     <tr>
99  *       <td>{@link javax.xml.XMLConstants#RELAXNG_NS_URI} ("<code>http://relaxng.org/ns/structure
/1.0</code>")</td>
100  *       <td><a href="http://www.relaxng.org/">RELAX NG 1.0</a></td>
101  *     </tr>
102  *   </tbody>
103  * </table>
104  *
105  * @author <a href="mailto:Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
106  * @author <a href="mailto:Neeraj.Bajaj@sun.com">Neeraj Bajaj</a>
107  *
108  * @since 1.5
109  */
110 public abstract class SchemaFactory {
111
112     private static SecuritySupport ss = new SecuritySupport();
113
114     /**
115     * <p>Constructor for derived classes.</p>
116     *
117     * <p>The constructor does nothing.</p>
118     *
119     * <p>Derived classes must create {@link SchemaFactory} objects that have
120     * <code>null</code> {@link ErrorHandler} and
121     * <code>null</code> {@link LSResourceResolver}.</p>
122     */

```

```

123     protected SchemaFactory() {
124     }
125
126     /**
127      * <p>Lookup an implementation of the SchemaFactory that supports the specified
128      * schema language and return it.</p>
129      *
130      * <p>To find a SchemaFactory object for a given schema language,
131      * this method looks the following places in the following order
132      * where "the class loader" refers to the context class loader:</p>
133      * <ol>
134      * <li>
135      *     If the system property
136      *     "javax.xml.validation.SchemaFactory:<i>schemaLanguage</i>"
137      *     is present (where <i>schemaLanguage</i> is the parameter
138      *     to this method), then its value is read
139      *     as a class name. The method will try to
140      *     create a new instance of this class by using the class loader,
141      *     and returns it if it is successfully created.
142      * </li>
143      * <li>
144      *     $java.home/lib/jaxp.properties is read and
145      *     the value associated with the key being the system property above
146      *     is looked for. If present, the value is processed just like above.
147      * </li>
148      * <li>
149      *     Use the service-provider loading facilities, defined by the
150      *     {@link java.util.ServiceLoader} class, to attempt to locate and load an
151      *     implementation of the service using the {@linkplain
152      *     java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
153      *     the service-provider loading facility will use the {@linkplain
154      *     java.lang.Thread#getContextClassLoader() current thread's context class loader}
155      *     to attempt to load the service. If the context class
156      *     loader is null, the {@linkplain
157      *     ClassLoader#getSystemClassLoader() system class loader} will be used.
158      * <br>
159      *     Each potential service provider is required to implement the method
160      *     {@link #isSchemaLanguageSupported(String schemaLanguage)}.
161      * <br>
162      *     The first service provider found that supports the specified schema
163      *     language is returned.
164      * <br>
165      *     In case of {@link java.util.ServiceConfigurationError} a
166      *     {@link SchemaFactoryConfigurationError} will be thrown.
167      * </li>
168      * <li>
169      *     Platform default SchemaFactory is located
170      *     in a implementation specific way. There must be a platform default
171      *     SchemaFactory for W3C XML Schema.
172      * </li>
173      * </ol>
174      *
175      * <p>If everything fails, {@link IllegalArgumentException} will be thrown.</p>

```

```

176 *
177 * <p><strong>Tip for Trouble-shooting:</strong></p>
178 * <p>See {@link java.util.Properties#load(java.io.InputStream)} for
179 * exactly how a property file is parsed. In particular, colons ':'
180 * need to be escaped in a property file, so make sure schema language
181 * URIs are properly escaped in it. For example:</p>
182 * <pre>
183 * http://www.w3.org/2001/XMLSchema=org.acme.foo.XSSchemaFactory
184 * </pre>
185 *
186 * @param schemaLanguage
187 *     Specifies the schema language which the returned
188 *     SchemaFactory will understand. See
189 *     <a href="#schemaLanguage">the list of available
190 *     schema languages</a> for the possible values.
191 *
192 * @return New instance of a <code>SchemaFactory</code>
193 *
194 * @throws IllegalArgumentException
195 *     If no implementation of the schema language is available.
196 * @throws NullPointerException
197 *     If the <code>schemaLanguage</code> parameter is null.
198 * @throws SchemaFactoryConfigurationError
199 *     If a configuration error is encountered.
200 *
201 * @see #newInstance(String schemaLanguage, String factoryClassName, ClassLoader classLoader)
202 */
203 public static SchemaFactory newInstance(String schemaLanguage) {
204     ClassLoader cl;
205     cl = ss.getContextClassLoader();
206
207     if (cl == null) {
208         //cl = ClassLoader.getSystemClassLoader();
209         //use the current class loader
210         cl = SchemaFactory.class.getClassLoader();
211     }
212
213     SchemaFactory f = new SchemaFactoryFinder(cl).newFactory(schemaLanguage);
214     if (f == null) {
215         throw new IllegalArgumentException(
216             "No SchemaFactory"
217             + " that implements the schema language specified by: " + schemaLanguage
218             + " could be loaded");
219     }
220     return f;
221 }
222
223 /**
224 * <p>Obtain a new instance of a <code>SchemaFactory</code> from class name. <code>
225 *     SchemaFactory</code>
226 * is returned if specified factory class name supports the specified schema language.
227 * This function is useful when there are multiple providers in the classpath.
228 * It gives more control to the application as it can specify which provider

```

```

228 * should be loaded.</p>
229 *
230 * <h2>Tip for Trouble-shooting</h2>
231 * <p>Setting the <code>jaxp.debug</code> system property will cause
232 * this method to print a lot of debug messages
233 * to <code>System.err</code> about what it is doing and where it is looking at.</p>
234 *
235 * <p> If you have problems try:</p>
236 * <pre>
237 * java -Djaxp.debug=1 YourProgram ....
238 * </pre>
239 *
240 * @param schemaLanguage Specifies the schema language which the returned
241 *           <code>SchemaFactory</code> will understand. See
242 *           <a href="#schemaLanguage">the list of available
243 *           schema languages</a> for the possible values.
244 *
245 * @param factoryClassName fully qualified factory class name that provides implementation of <
246 *           code>javax.xml.validation.SchemaFactory</code>.
247 *
248 * @param classLoader <code>ClassLoader</code> used to load the factory class. If <code>null</
249 *           code>
250 *           current <code>Thread</code>'s context classLoader is used to load the
251 *           factory class.
252 *
253 * @return New instance of a <code>SchemaFactory</code>
254 *
255 * @throws IllegalArgumentException
256 *           if <code>factoryClassName</code> is <code>null</code>, or
257 *           the factory class cannot be loaded, instantiated or doesn't
258 *           support the schema language specified in <code>schemLanguage</code>
259 *           parameter.
260 *
261 * @throws NullPointerException
262 *           If the <code>schemaLanguage</code> parameter is null.
263 *
264 * @see #newInstance(String schemaLanguage)
265 *
266 * @since 1.6
267 */
268 public static SchemaFactory newInstance(String schemaLanguage, String factoryClassName,
269                                       ClassLoader classLoader){
270     ClassLoader cl = classLoader;
271
272     if (cl == null) {
273         cl = ss.getContextClassLoader();
274     }
275
276     SchemaFactory f = new SchemaFactoryFinder(cl).createInstance(factoryClassName);
277     if (f == null) {
278         throw new IllegalArgumentException(
279             "Factory " + factoryClassName
280             + " could not be loaded to implement the schema language specified by: " +

```

```

        schemaLanguage);
277     }
278     //if this factory supports the given schemalanguage return this factory else thrown
        exception
279     if(f.isSchemaLanguageSupported(schemaLanguage)){
280         return f;
281     }else{
282         throw new IllegalArgumentException(
283             "Factory " + f.getClass().getName()
284             + " does not implement the schema language specified by: " + schemaLanguage);
285     }
286
287 }
288
289 /**
290  * <p>Is specified schema supported by this <code>SchemaFactory</code>?</p>
291  *
292  * @param schemaLanguage Specifies the schema language which the returned <code>SchemaFactory</code>
        code> will understand.
293  * <code>schemaLanguage</code> must specify a <a href="#schemaLanguage">valid</a> schema
        language.
294  *
295  * @return <code>true</code> if <code>SchemaFactory</code> supports <code>schemaLanguage</code>
        >, else <code>false</code>.
296  *
297  * @throws NullPointerException If <code>schemaLanguage</code> is <code>null</code>.
298  * @throws IllegalArgumentException If <code>schemaLanguage.length() == 0</code>
299  * or <code>schemaLanguage</code> does not specify a <a href="#schemaLanguage">valid</a>
        schema language.
300  */
301 public abstract boolean isSchemaLanguageSupported(String schemaLanguage);
302
303 /**
304  * Look up the value of a feature flag.
305  *
306  * <p>The feature name is any fully-qualified URI. It is
307  * possible for a {@link SchemaFactory} to recognize a feature name but
308  * temporarily be unable to return its value.
309  *
310  * <p>Implementors are free (and encouraged) to invent their own features,
311  * using names built on their own URIs.</p>
312  *
313  * @param name The feature name, which is a non-null fully-qualified URI.
314  *
315  * @return The current value of the feature (true or false).
316  *
317  * @throws SAXNotRecognizedException If the feature
318  * value can't be assigned or retrieved.
319  * @throws SAXNotSupportedException When the
320  * {@link SchemaFactory} recognizes the feature name but
321  * cannot determine its value at this time.
322  * @throws NullPointerException If <code>name</code> is <code>null</code>.
323  */

```

```

324     * @see #setFeature(String, boolean)
325     */
326     public boolean getFeature(String name)
327         throws SAXNotRecognizedException, SAXNotSupportedException {
328
329         if (name == null) {
330             throw new NullPointerException("the name parameter is null");
331         }
332         throw new SAXNotRecognizedException(name);
333     }
334
335     /**
336     * <p>Set a feature for this <code>SchemaFactory</code>,
337     * {@link Schema}s created by this factory, and by extension,
338     * {@link Validator}s and {@link ValidatorHandler}s created by
339     * those {@link Schema}s.
340     * </p>
341     *
342     * <p>Implementors and developers should pay particular attention
343     * to how the special {@link Schema} object returned by {@link
344     * #newSchema()} is processed. In some cases, for example, when the
345     * <code>SchemaFactory</code> and the class actually loading the
346     * schema come from different implementations, it may not be possible
347     * for <code>SchemaFactory</code> features to be inherited automatically.
348     * Developers should
349     * make sure that features, such as secure processing, are explicitly
350     * set in both places.</p>
351     *
352     * <p>The feature name is any fully-qualified URI. It is
353     * possible for a {@link SchemaFactory} to expose a feature value but
354     * to be unable to change the current value.</p>
355     *
356     * <p>All implementations are required to support the {@link javax.xml.XMLConstants#
357     *     FEATURE_SECURE_PROCESSING} feature.
358     * When the feature is:</p>
359     * <ul>
360     *     <li>
361     *         <code>true</code>: the implementation will limit XML processing to conform to
362     *         implementation limits.
363     *         Examples include entity expansion limits and XML Schema constructs that would consume
364     *         large amounts of resources.
365     *         If XML processing is limited for security reasons, it will be reported via a call to the
366     *         registered
367     *         {@link ErrorHandler#fatalError(SAXParseException exception)}.
368     *         See {@link #setErrorHandler(ErrorHandler errorHandler)}.
369     *     </li>
370     *     <li>
371     *         <code>false</code>: the implementation will processing XML according to the XML
372     *         specifications without
373     *         regard to possible implementation limits.
374     *     </li>
375     * </ul>
376     *

```



```

372 * @param name The feature name, which is a non-null fully-qualified URI.
373 * @param value The requested value of the feature (true or false).
374 *
375 * @throws SAXNotRecognizedException If the feature
376 *     value can't be assigned or retrieved.
377 * @throws SAXNotSupportedException When the
378 *     {@link SchemaFactory} recognizes the feature name but
379 *     cannot set the requested value.
380 * @throws NullPointerException If name is null.
381 *
382 * @see #getFeature(String)
383 */
384 public void setFeature(String name, boolean value)
385     throws SAXNotRecognizedException, SAXNotSupportedException {
386
387     if (name == null) {
388         throw new NullPointerException("the name parameter is null");
389     }
390     throw new SAXNotRecognizedException(name);
391 }
392
393 /**
394  * Set the value of a property.
395  *
396  * <p>The property name is any fully-qualified URI. It is
397  * possible for a {@link SchemaFactory} to recognize a property name but
398  * to be unable to change the current value.</p>
399  *
400  * <p>
401  * All implementations that implement JAXP 1.5 or newer are required to
402  * support the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} and
403  * {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} properties.
404  * </p>
405  * <ul>
406  * <li>
407  * <p>Access to external DTDs in Schema files is restricted to the protocols
408  * specified by the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} property.
409  * If access is denied during the creation of new Schema due to the restriction
410  * of this property, {@link org.xml.sax.SAXException} will be thrown by the
411  * {@link #newSchema(Source)} or {@link #newSchema(File)}
412  * or {@link #newSchema(URL)} or or {@link #newSchema(Source[])} method.</p>
413  *
414  * <p>Access to external DTDs in xml source files is restricted to the protocols
415  * specified by the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} property.
416  * If access is denied during validation due to the restriction
417  * of this property, {@link org.xml.sax.SAXException} will be thrown by the
418  * {@link javax.xml.validation.Validator#validate(Source)} or
419  * {@link javax.xml.validation.Validator#validate(Source, Result)} method.</p>
420  *
421  * <p>Access to external reference set by the schemaLocation attribute is
422  * restricted to the protocols specified by the
423  * {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} property.
424  * If access is denied during validation due to the restriction of this property,

```

```

425 *      {@link org.xml.sax.SAXException} will be thrown by the
426 *      {@link javax.xml.validation.Validator#validate(Source)} or
427 *      {@link javax.xml.validation.Validator#validate(Source, Result)} method.</p>
428 *
429 *      <p>Access to external reference set by the Import
430 *      and Include element is restricted to the protocols specified by the
431 *      {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} property.
432 *      If access is denied during the creation of new Schema due to the restriction
433 *      of this property, {@link org.xml.sax.SAXException} will be thrown by the
434 *      {@link #newSchema(Source)} or {@link #newSchema(File)}
435 *      or {@link #newSchema(URL)} or {@link #newSchema(Source[])} method.</p>
436 *    </li>
437 * </ul>
438 *
439 * @param name The property name, which is a non-null fully-qualified URI.
440 * @param object The requested value for the property.
441 *
442 * @throws SAXNotRecognizedException If the property
443 *     value can't be assigned or retrieved.
444 * @throws SAXNotSupportedException When the
445 *     {@link SchemaFactory} recognizes the property name but
446 *     cannot set the requested value.
447 * @throws NullPointerException If <code>name</code> is <code>null</code>.
448 */
449 public void setProperty(String name, Object object)
450     throws SAXNotRecognizedException, SAXNotSupportedException {
451
452     if (name == null) {
453         throw new NullPointerException("the name parameter is null");
454     }
455     throw new SAXNotRecognizedException(name);
456 }
457
458 /**
459 * Look up the value of a property.
460 *
461 * <p>The property name is any fully-qualified URI. It is
462 * possible for a {@link SchemaFactory} to recognize a property name but
463 * temporarily be unable to return its value.</p>
464 *
465 * <p>{@link SchemaFactory}s are not required to recognize any specific
466 * property names.</p>
467 *
468 * <p>Implementors are free (and encouraged) to invent their own properties,
469 * using names built on their own URIs.</p>
470 *
471 * @param name The property name, which is a non-null fully-qualified URI.
472 *
473 * @return The current value of the property.
474 *
475 * @throws SAXNotRecognizedException If the property
476 *     value can't be assigned or retrieved.
477 * @throws SAXNotSupportedException When the

```

```

478 * XMLReader recognizes the property name but
479 * cannot determine its value at this time.
480 * @throws NullPointerException If <code>name</code> is <code>null</code>.
481 *
482 * @see #setProperty(String, Object)
483 */
484 public Object getProperty(String name)
485     throws SAXNotRecognizedException, SAXNotSupportedException {
486
487     if (name == null) {
488         throw new NullPointerException("the name parameter is null");
489     }
490     throw new SAXNotRecognizedException(name);
491 }
492
493 /**
494 * Sets the {@link ErrorHandler} to receive errors encountered
495 * during the <code>newSchema</code> method invocation.
496 *
497 * <p>
498 * Error handler can be used to customize the error handling process
499 * during schema parsing. When an {@link ErrorHandler} is set,
500 * errors found during the parsing of schemas will be first sent
501 * to the {@link ErrorHandler}.
502 *
503 * <p>
504 * The error handler can abort the parsing of a schema immediately
505 * by throwing {@link SAXException} from the handler. Or for example
506 * it can print an error to the screen and try to continue the
507 * processing by returning normally from the {@link ErrorHandler}
508 *
509 * <p>
510 * If any {@link Throwable} (or instances of its derived classes)
511 * is thrown from an {@link ErrorHandler},
512 * the caller of the <code>newSchema</code> method will be thrown
513 * the same {@link Throwable} object.
514 *
515 * <p>
516 * {@link SchemaFactory} is not allowed to
517 * throw {@link SAXException} without first reporting it to
518 * {@link ErrorHandler}.
519 *
520 * <p>
521 * Applications can call this method even during a {@link Schema}
522 * is being parsed.
523 *
524 * <p>
525 * When the {@link ErrorHandler} is null, the implementation will
526 * behave as if the following {@link ErrorHandler} is set:
527 * <pre>
528 * class DraconianErrorHandler implements {@link ErrorHandler} {
529 *     public void fatalError( {@link org.xml.sax.SAXParseException} e ) throws {@link
530         SAXException} {

```

```

530     *         throw e;
531     *     }
532     *     public void error( {@link org.xml.sax.SAXParseException} e ) throws {@link SAXException}
533     *     {
534     *         throw e;
535     *     }
536     *     public void warning( {@link org.xml.sax.SAXParseException} e ) throws {@link
537     *         SAXException} {
538     *         // noop
539     *     }
540     * }
541     * </pre>
542     * <p>
543     * When a new {@link SchemaFactory} object is created, initially
544     * this field is set to null. This field will <em>NOT</em> be
545     * inherited to {@link Schema}s, {@link Validator}s, or
546     * {@link ValidatorHandler}s that are created from this {@link SchemaFactory}.
547     * @param errorHandler A new error handler to be set.
548     * This parameter can be <code>null</code>.
549     */
550     public abstract void setErrorHandler(ErrorHandler errorHandler);
551
552     /**
553     * Gets the current {@link ErrorHandler} set to this {@link SchemaFactory}.
554     *
555     * @return
556     * This method returns the object that was last set through
557     * the {@link #setErrorHandler(ErrorHandler)} method, or null
558     * if that method has never been called since this {@link SchemaFactory}
559     * has created.
560     *
561     * @see #setErrorHandler(ErrorHandler)
562     */
563     public abstract ErrorHandler getErrorHandler();
564
565     /**
566     * Sets the {@link LSResourceResolver} to customize
567     * resource resolution when parsing schemas.
568     *
569     * <p>
570     * {@link SchemaFactory} uses a {@link LSResourceResolver}
571     * when it needs to locate external resources while parsing schemas,
572     * although exactly what constitutes "locating external resources" is
573     * up to each schema language. For example, for W3C XML Schema,
574     * this includes files <code>&lt;include></code>d or <code>&lt;import></code>ed,
575     * and DTD referenced from schema files, etc.
576     *
577     * <p>
578     * Applications can call this method even during a {@link Schema}
579     * is being parsed.
580     *

```

```

581 * <p>
582 * When the {@link LSResourceResolver} is null, the implementation will
583 * behave as if the following {@link LSResourceResolver} is set:
584 * <pre>
585 * class DumbDOMResourceResolver implements {@link LSResourceResolver} {
586 *     public {@link org.w3c.dom.ls.LSInput} resolveResource(
587 *         String publicId, String systemId, String baseURI) {
588 *
589 *         return null; // always return null
590 *     }
591 * }
592 * </pre>
593 *
594 * <p>
595 * If a {@link LSResourceResolver} throws a {@link RuntimeException}
596 * (or instances of its derived classes),
597 * then the {@link SchemaFactory} will abort the parsing and
598 * the caller of the <code>newSchema</code> method will receive
599 * the same {@link RuntimeException}.
600 *
601 * <p>
602 * When a new {@link SchemaFactory} object is created, initially
603 * this field is set to null. This field will <em>NOT</em> be
604 * inherited to {@link Schema}s, {@link Validator}s, or
605 * {@link ValidatorHandler}s that are created from this {@link SchemaFactory}.
606 *
607 * @param resourceResolver
608 *     A new resource resolver to be set. This parameter can be null.
609 */
610 public abstract void setResourceResolver(LSResourceResolver resourceResolver);
611
612 /**
613 * Gets the current {@link LSResourceResolver} set to this {@link SchemaFactory}.
614 *
615 * @return
616 *     This method returns the object that was last set through
617 *     the {@link #setResourceResolver(LSResourceResolver)} method, or null
618 *     if that method has never been called since this {@link SchemaFactory}
619 *     has created.
620 *
621 * @see #setErrorHandler(ErrorHandler)
622 */
623 public abstract LSResourceResolver getResourceResolver();
624
625 /**
626 * <p>Parses the specified source as a schema and returns it as a schema.</p>
627 *
628 * <p>This is a convenience method for {@link #newSchema(Source[] schemas)}.</p>
629 *
630 * @param schema Source that represents a schema.
631 *
632 * @return New <code>Schema</code> from parsing <code>schema</code>.
633 *

```

```

634     * @throws SAXException If a SAX error occurs during parsing.
635     * @throws NullPointerException if <code>schema</code> is null.
636     */
637     public Schema newSchema(Source schema) throws SAXException {
638         return newSchema(new Source[]{schema});
639     }
640
641     /**
642     * <p>Parses the specified <code>File</code> as a schema and returns it as a <code>Schema</code>
643     * </p>
644     * <p>This is a convenience method for {@link #newSchema(Source schema)}.</p>
645     *
646     * @param schema File that represents a schema.
647     *
648     * @return New <code>Schema</code> from parsing <code>schema</code>.
649     *
650     * @throws SAXException If a SAX error occurs during parsing.
651     * @throws NullPointerException if <code>schema</code> is null.
652     */
653     public Schema newSchema(File schema) throws SAXException {
654         return newSchema(new StreamSource(schema));
655     }
656
657     /**
658     * <p>Parses the specified <code>URL</code> as a schema and returns it as a <code>Schema</code>
659     * </p>
660     * <p>This is a convenience method for {@link #newSchema(Source schema)}.</p>
661     *
662     * @param schema <code>URL</code> that represents a schema.
663     *
664     * @return New <code>Schema</code> from parsing <code>schema</code>.
665     *
666     * @throws SAXException If a SAX error occurs during parsing.
667     * @throws NullPointerException if <code>schema</code> is null.
668     */
669     public Schema newSchema(URL schema) throws SAXException {
670         return newSchema(new StreamSource(schema.toExternalForm()));
671     }
672
673     /**
674     * Parses the specified source(s) as a schema and returns it as a schema.
675     *
676     * <p>
677     * The callee will read all the {@link Source}s and combine them into a
678     * single schema. The exact semantics of the combination depends on the schema
679     * language that this {@link SchemaFactory} object is created for.
680     *
681     * <p>
682     * When an {@link ErrorHandler} is set, the callee will report all the errors
683     * found in sources to the handler. If the handler throws an exception, it will
684     * abort the schema compilation and the same exception will be thrown from

```

```

685 * this method. Also, after an error is reported to a handler, the callee is allowed
686 * to abort the further processing by throwing it. If an error handler is not set,
687 * the callee will throw the first error it finds in the sources.
688 *
689 * <h2>W3C XML Schema 1.0</h2>
690 * <p>
691 * The resulting schema contains components from the specified sources.
692 * The same result would be achieved if all these sources were
693 * imported, using appropriate values for schemaLocation and namespace,
694 * into a single schema document with a different targetNamespace
695 * and no components of its own, if the import elements were given
696 * in the same order as the sources. Section 4.2.3 of the XML Schema
697 * recommendation describes the options processors have in this
698 * regard. While a processor should be consistent in its treatment of
699 * JAXP schema sources and XML Schema imports, the behaviour between
700 * JAXP-compliant parsers may vary; in particular, parsers may choose
701 * to ignore all but the first <import> for a given namespace,
702 * regardless of information provided in schemaLocation.
703 *
704 * <p>
705 * If the parsed set of schemas includes error(s) as
706 * specified in the section 5.1 of the XML Schema spec, then
707 * the error must be reported to the {@link ErrorHandler}.
708 *
709 * <h2>RELAX NG</h2>
710 *
711 * <p>For RELAX NG, this method must throw {@link UnsupportedOperationException}
712 * if <code>schemas.length!=1</code>.
713 *
714 *
715 * @param schemas
716 *     inputs to be parsed. {@link SchemaFactory} is required
717 *     to recognize {@link javax.xml.transform.sax.SAXSource},
718 *     {@link StreamSource},
719 *     {@link javax.xml.transform.stax.StAXSource},
720 *     and {@link javax.xml.transform.dom.DOMSource}.
721 *     Input schemas must be XML documents or
722 *     XML elements and must not be null. For backwards compatibility,
723 *     the results of passing anything other than
724 *     a document or element are implementation-dependent.
725 *     Implementations must either recognize and process the input
726 *     or throw an IllegalArgumentException.
727 *
728 * @return
729 *     Always return a non-null valid {@link Schema} object.
730 *     Note that when an error has been reported, there is no
731 *     guarantee that the returned {@link Schema} object is
732 *     meaningful.
733 *
734 * @throws SAXException
735 *     If an error is found during processing the specified inputs.
736 *     When an {@link ErrorHandler} is set, errors are reported to
737 *     there first. See {@link #setErrorHandler(ErrorHandler)}.

```

```

738     * @throws NullPointerException
739     *     If the <code>schemas</code> parameter itself is null or
740     *     any item in the array is null.
741     * @throws IllegalArgumentException
742     *     If any item in the array is not recognized by this method.
743     * @throws UnsupportedOperationException
744     *     If the schema language doesn't support this operation.
745     */
746     public abstract Schema newSchema(Source[] schemas) throws SAXException;
747
748     /**
749     * Creates a special {@link Schema} object.
750     *
751     * <p>The exact semantics of the returned {@link Schema} object
752     * depend on the schema language for which this {@link SchemaFactory}
753     * is created.
754     *
755     * <p>Also, implementations are allowed to use implementation-specific
756     * property/feature to alter the semantics of this method.</p>
757     *
758     * <p>Implementors and developers should pay particular attention
759     * to how the features set on this {@link SchemaFactory} are
760     * processed by this special {@link Schema}.
761     * In some cases, for example, when the
762     * {@link SchemaFactory} and the class actually loading the
763     * schema come from different implementations, it may not be possible
764     * for {@link SchemaFactory} features to be inherited automatically.
765     * Developers should
766     * make sure that features, such as secure processing, are explicitly
767     * set in both places.</p>
768     *
769     * <h2>W3C XML Schema 1.0</h2>
770     * <p>
771     * For XML Schema, this method creates a {@link Schema} object that
772     * performs validation by using location hints specified in documents.
773     *
774     * <p>
775     * The returned {@link Schema} object assumes that if documents
776     * refer to the same URL in the schema location hints,
777     * they will always resolve to the same schema document. This
778     * assumption allows implementations to reuse parsed results of
779     * schema documents so that multiple validations against the same
780     * schema will run faster.
781     *
782     * <p>
783     * Note that the use of schema location hints introduces a
784     * vulnerability to denial-of-service attacks.
785     *
786     *
787     * <h2>RELAX NG</h2>
788     * <p>
789     * RELAX NG does not support this operation.
790     *

```



```

791     * @return
792     *     Always return non-null valid {@link Schema} object.
793     *
794     * @throws UnsupportedOperationException
795     *     If this operation is not supported by the callee.
796     * @throws SAXException
797     *     If this operation is supported but failed for some reason.
798     */
799     public abstract Schema newSchema() throws SAXException;
800 }

```

## 26.3 SchemaFactoryConfigurationError.java

Secuencia 253: javax/xml/validation/SchemaFactoryConfigurationError.java

```

1  /*
2  * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.validation;
27
28 /**
29  * Thrown when a problem with configuration with the Schema Factories
30  * exists. This error will typically be thrown when the class of a
31  * schema factory specified in the system properties cannot be found
32  * or instantiated.
33  * @since 1.8
34  */
35 public final class SchemaFactoryConfigurationError extends Error {
36
37     static final long serialVersionUID = 3531438703147750126L;
38
39     /**

```

```

40     * Create a new <code>SchemaFactoryConfigurationError</code> with no
41     * detail message.
42     */
43     public SchemaFactoryConfigurationError() {
44     }
45
46
47     /**
48     * Create a new <code>SchemaFactoryConfigurationError</code> with
49     * the <code>String</code> specified as an error message.
50     *
51     * @param message The error message for the exception.
52     */
53     public SchemaFactoryConfigurationError(String message) {
54         super(message);
55     }
56
57     /**
58     * Create a new <code>SchemaFactoryConfigurationError</code> with the
59     * given <code>Throwable</code> base cause.
60     *
61     * @param cause The exception or error to be encapsulated in a
62     * SchemaFactoryConfigurationError.
63     */
64     public SchemaFactoryConfigurationError(Throwable cause) {
65         super(cause);
66     }
67
68     /**
69     * Create a new <code>SchemaFactoryConfigurationError</code> with the
70     * given <code>Throwable</code> base cause and detail message.
71     *
72     * @param cause The exception or error to be encapsulated in a
73     * SchemaFactoryConfigurationError.
74     * @param message The detail message.
75     */
76     public SchemaFactoryConfigurationError(String message, Throwable cause) {
77         super(message, cause);
78     }
79
80 }

```

## 26.4 SchemaFactoryFinder.java

Secuencia 254: javax/xml/validation/SchemaFactoryFinder.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.validation;
27
28 import java.io.File;
29 import java.lang.reflect.Method;
30 import java.lang.reflect.Modifier;
31 import java.net.URL;
32 import java.security.AccessControlContext;
33 import java.security.AccessController;
34 import java.security.PrivilegedAction;
35 import java.util.Properties;
36 import java.util.ServiceConfigurationError;
37 import java.util.ServiceLoader;
38
39 /**
40  * Implementation of {@link SchemaFactory#newInstance(String)}.
41  *
42  * @author <a href="Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
43  * @version $Revision: 1.8 $, $Date: 2010-11-01 04:36:13 $
44  * @since 1.5
45  */
46 class SchemaFactoryFinder {
47
48     /** debug support code. */
49     private static boolean debug = false;
50
51     /**
52      * <p> Take care of restrictions imposed by java security model </p>
53      */
54     private static final SecuritySupport ss = new SecuritySupport();
55     private static final String DEFAULT_PACKAGE = "com.sun.org.apache.xerces.internal";
56
57     /**
58      * <p>Cache properties for performance.</p>
59      */
60     private static final Properties cacheProps = new Properties();
61
62     /**
63      * <p>First time requires initialization overhead.</p>
64      */
65 }
```

```

62     */
63     private static volatile boolean firstTime = true;
64
65     static {
66         // Use try/catch block to support applets
67         try {
68             debug = ss.getProperty("jaxp.debug") != null;
69         } catch (Exception unused) {
70             debug = false;
71         }
72     }
73
74     /**
75      * <p>Conditional debug printing.</p>
76      *
77      * @param msg to print
78      */
79     private static void debugPrintln(String msg) {
80         if (debug) {
81             System.err.println("JAXP: " + msg);
82         }
83     }
84
85     /**
86      * <p><code>ClassLoader</code> to use to find <code>SchemaFactory</code>.</p>
87      */
88     private final ClassLoader classLoader;
89
90     /**
91      * <p>Constructor that specifies <code>ClassLoader</code> to use
92      * to find <code>SchemaFactory</code>.</p>
93      *
94      * @param loader
95      *     to be used to load resource, {@link SchemaFactory}, and
96      *     {@link SchemaFactoryLoader} implementations during
97      *     the resolution process.
98      *     If this parameter is null, the default system class loader
99      *     will be used.
100     */
101     public SchemaFactoryFinder(ClassLoader loader) {
102         this.classLoader = loader;
103         if (debug) {
104             debugDisplayClassLoader();
105         }
106     }
107
108     private void debugDisplayClassLoader() {
109         try {
110             if (classLoader == ss.getContextClassLoader()) {
111                 debugPrintln("using thread context class loader (" + classLoader + ") for search");
112                 return;
113             }
114         } catch (Throwable unused) {

```

```

115     // getContextClassLoader() undefined in JDK1.1
116     }
117
118     if( classLoader==ClassLoader.getSystemClassLoader() ) {
119         debugPrintln("using system class loader (" +classLoader+") for search");
120         return;
121     }
122
123     debugPrintln("using class loader (" +classLoader+") for search");
124 }
125
126 /**
127  * <p>Creates a new {@link SchemaFactory} object for the specified
128  * schema language.</p>
129  *
130  * @param schemaLanguage
131  *     See {@link SchemaFactory Schema Language} table in <code>SchemaFactory</code>
132  *     for the list of available schema languages.
133  *
134  * @return <code>null</code> if the callee fails to create one.
135  *
136  * @throws NullPointerException
137  *     If the <code>schemaLanguage</code> parameter is null.
138  * @throws SchemaFactoryConfigurationException
139  *     If a configuration error is encountered.
140  */
141 public SchemaFactory newFactory(String schemaLanguage) {
142     if(schemaLanguage==null) {
143         throw new NullPointerException();
144     }
145     SchemaFactory f = _newFactory(schemaLanguage);
146     if (f != null) {
147         debugPrintln("factory '" + f.getClass().getName() + "' was found for " + schemaLanguage
148             );
149     } else {
150         debugPrintln("unable to find a factory for " + schemaLanguage);
151     }
152     return f;
153 }
154
155 /**
156  * <p>Lookup a <code>SchemaFactory</code> for the given <code>schemaLanguage</code>.</p>
157  *
158  * @param schemaLanguage Schema language to lookup <code>SchemaFactory</code> for.
159  *
160  * @return <code>SchemaFactory</code> for the given <code>schemaLanguage</code>.
161  */
162 private SchemaFactory _newFactory(String schemaLanguage) {
163     SchemaFactory sf;
164
165     String propertyName = SERVICE_CLASS.getName() + "." + schemaLanguage;
166
167     // system property look up

```

```

167     try {
168         debugPrintln("Looking up system property '"+propertyName+"' ");
169         String r = ss.getSystemProperty(propertyName);
170         if(r!=null) {
171             debugPrintln("The value is '"+r+"'");
172             sf = createInstance(r, true);
173             if(sf!=null) return sf;
174         } else
175             debugPrintln("The property is undefined.");
176     } catch( Throwable t ) {
177         if( debug ) {
178             debugPrintln("failed to look up system property '"+propertyName+"' ");
179             t.printStackTrace();
180         }
181     }
182
183     String javah = ss.getSystemProperty( "java.home" );
184     String configFile = javah + File.separator +
185     "lib" + File.separator + "jaxp.properties";
186
187
188     // try to read from $java.home/lib/jaxp.properties
189     try {
190         if(firstTime){
191             synchronized(cacheProps){
192                 if(firstTime){
193                     File f=new File( configFile );
194                     firstTime = false;
195                     if(ss.doesFileExist(f)){
196                         debugPrintln("Read properties file " + f);
197                         cacheProps.load(ss.getFileInputStream(f));
198                     }
199                 }
200             }
201         }
202         final String factoryClassName = cacheProps.getProperty(propertyName);
203         debugPrintln("found " + factoryClassName + " in $java.home/jaxp.properties");
204
205         if (factoryClassName != null) {
206             sf = createInstance(factoryClassName, true);
207             if(sf != null){
208                 return sf;
209             }
210         }
211     } catch (Exception ex) {
212         if (debug) {
213             ex.printStackTrace();
214         }
215     }
216
217     // Try with ServiceLoader
218     final SchemaFactory factoryImpl = findServiceProvider(schemaLanguage);
219

```

```
220     // The following assertion should always be true.
221     // Uncomment it, recompile, and run with -ea in case of doubts:
222     // assert factoryImpl == null || factoryImpl.isSchemaLanguageSupported(schemaLanguage);
223
224     if (factoryImpl != null) {
225         return factoryImpl;
226     }
227
228     // platform default
229     if (schemaLanguage.equals("http://www.w3.org/2001/XMLSchema")) {
230         debugPrintln("attempting to use the platform default XML Schema validator");
231         return createInstance("com.sun.org.apache.xerces.internal.jaxp.validation.
232             XMLSchemaFactory", true);
233     }
234
235     debugPrintln("all things were tried, but none was found. bailing out.");
236     return null;
237 }
238
239 /** <p>Create class using appropriate ClassLoader.</p>
240  *
241  * @param className Name of class to create.
242  * @return Created class or <code>null</code>.
243  */
244 private Class<?> createClass(String className) {
245     Class<?> clazz;
246     // make sure we have access to restricted packages
247     boolean internal = false;
248     if (System.getSecurityManager() != null) {
249         if (className != null && className.startsWith(DEFAULT_PACKAGE)) {
250             internal = true;
251         }
252     }
253
254     try {
255         if (classLoader != null && !internal) {
256             clazz = Class.forName(className, false, classLoader);
257         } else {
258             clazz = Class.forName(className);
259         }
260     } catch (Throwable t) {
261         if (debug) {
262             t.printStackTrace();
263         }
264         return null;
265     }
266
267     return clazz;
268 }
269
270 /**
271  * <p>Creates an instance of the specified and returns it.</p>
272  *
```

```

272     * @param className
273     *     fully qualified class name to be instantiated.
274     *
275     * @return null
276     *     if it fails. Error messages will be printed by this method.
277     */
278     SchemaFactory createInstance( String className ) {
279         return createInstance( className, false );
280     }
281
282     SchemaFactory createInstance( String className, boolean useServicesMechanism ) {
283         SchemaFactory schemaFactory = null;
284
285         debugPrintln("createInstance(" + className + ")");
286
287         // get Class from className
288         Class<?> clazz = createClass(className);
289         if (clazz == null) {
290             debugPrintln("failed to getClass(" + className + ")");
291             return null;
292         }
293         debugPrintln("loaded " + className + " from " + which(clazz));
294
295         // instantiate Class as a SchemaFactory
296         try {
297             if (!SchemaFactory.class.isAssignableFrom(clazz)) {
298                 throw new ClassCastException(clazz.getName()
299                     + " cannot be cast to " + SchemaFactory.class);
300             }
301             if (!useServicesMechanism) {
302                 schemaFactory = newInstanceNoServiceLoader(clazz);
303             }
304             if (schemaFactory == null) {
305                 schemaFactory = (SchemaFactory) clazz.newInstance();
306             }
307         } catch (ClassCastException classCastException) {
308             debugPrintln("could not instantiate " + clazz.getName());
309             if (debug) {
310                 classCastException.printStackTrace();
311             }
312             return null;
313         } catch (IllegalAccessException illegalAccessException) {
314             debugPrintln("could not instantiate " + clazz.getName());
315             if (debug) {
316                 illegalAccessException.printStackTrace();
317             }
318             return null;
319         } catch (InstantiationException instantiationException) {
320             debugPrintln("could not instantiate " + clazz.getName());
321             if (debug) {
322                 instantiationException.printStackTrace();
323             }
324             return null;

```



```

325     }
326
327     return schemaFactory;
328 }
329
330 /**
331  * Try to construct using newXMLSchemaFactoryNoServiceLoader
332  * method if available.
333  */
334 private static SchemaFactory newInstanceNoServiceLoader(
335     Class<?> providerClass
336 ) {
337     // Retain maximum compatibility if no security manager.
338     if (System.getSecurityManager() == null) {
339         return null;
340     }
341     try {
342         final Method creationMethod =
343             providerClass.getDeclaredMethod(
344                 "newXMLSchemaFactoryNoServiceLoader"
345             );
346         final int modifiers = creationMethod.getModifiers();
347
348         // Do not call the method if it's not public static.
349         if (!Modifier.isStatic(modifiers) || !Modifier.isPublic(modifiers)) {
350             return null;
351         }
352
353         // Only calls "newXMLSchemaFactoryNoServiceLoader" if it's
354         // declared to return an instance of SchemaFactory.
355         final Class<?> returnType = creationMethod.getReturnType();
356         if (SERVICE_CLASS.isAssignableFrom(returnType)) {
357             return SERVICE_CLASS.cast(creationMethod.invoke(null, (Object[])null));
358         } else {
359             // Should not happen since
360             // XMLSchemaFactory.newXMLSchemaFactoryNoServiceLoader is
361             // declared to return XMLSchemaFactory.
362             throw new ClassCastException(returnType
363                 + " cannot be cast to " + SERVICE_CLASS);
364         }
365     } catch (ClassCastException e) {
366         throw new SchemaFactoryConfigurationError(e.getMessage(), e);
367     } catch (NoSuchMethodException exc) {
368         return null;
369     } catch (Exception exc) {
370         return null;
371     }
372 }
373
374 // Call isSchemaLanguageSupported with initial context.
375 private boolean isSchemaLanguageSupportedBy(final SchemaFactory factory,
376     final String schemaLanguage,
377     AccessControlContext acc) {

```

```

378     return AccessController.doPrivileged(new PrivilegedAction<Boolean>() {
379         public Boolean run() {
380             return factory.isSchemaLanguageSupported(schemaLanguage);
381         }
382     }, acc);
383 }
384
385 /**
386  * Finds a service provider subclass of SchemaFactory that supports the
387  * given schema language using the ServiceLoader.
388  *
389  * @param schemaLanguage The schema language for which we seek a factory.
390  * @return A SchemaFactory supporting the specified schema language, or null
391  *         if none is found.
392  * @throws SchemaFactoryConfigurationError if a configuration error is found.
393  */
394 private SchemaFactory findServiceProvider(final String schemaLanguage) {
395     assert schemaLanguage != null;
396     // store current context.
397     final AccessControlContext acc = AccessController.getContext();
398     try {
399         return AccessController.doPrivileged(new PrivilegedAction<SchemaFactory>() {
400             public SchemaFactory run() {
401                 final ServiceLoader<SchemaFactory> loader =
402                     ServiceLoader.load(SERVICE_CLASS);
403                 for (SchemaFactory factory : loader) {
404                     // restore initial context to call
405                     // factory.isSchemaLanguageSupported
406                     if (isSchemaLanguageSupportedBy(factory, schemaLanguage, acc)) {
407                         return factory;
408                     }
409                 }
410                 return null; // no factory found.
411             }
412         });
413     } catch (ServiceConfigurationError error) {
414         throw new SchemaFactoryConfigurationError(
415             "Provider for " + SERVICE_CLASS + " cannot be created", error);
416     }
417 }
418
419 private static final Class<SchemaFactory> SERVICE_CLASS = SchemaFactory.class;
420
421
422 private static String which( Class<?> clazz ) {
423     return which( clazz.getName(), clazz.getClassLoader() );
424 }
425
426 /**
427  * <p>Search the specified classloader for the given classname.</p>
428  *
429  * @param classname the fully qualified name of the class to search for
430  * @param loader the classloader to search

```

```

431     *
432     * @return the source location of the resource, or null if it wasn't found
433     */
434     private static String which(String classname, ClassLoader loader) {
435
436         String classnameAsResource = classname.replace('.', '/') + ".class";
437
438         if( loader==null ) loader = ClassLoader.getSystemClassLoader();
439
440         //URL it = loader.getResource(classnameAsResource);
441         URL it = ss.getResourceAsURL(loader, classnameAsResource);
442         if (it != null) {
443             return it.toString();
444         } else {
445             return null;
446         }
447     }
448 }

```

## 26.5 SchemaFactoryLoader.java

Secuencia 255: javax/xml/validation/SchemaFactoryLoader.java

```

1  /*
2  * Copyright (c) 2004, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.validation;
27
28 /**
29  * <p>Factory that creates {@link SchemaFactory}</p>
30  *
31  * <p><b>DO NOT USE THIS CLASS</b></p>

```

```

32  *
33  * <p>
34  * This class was introduced as a part of an early proposal during the
35  * JSR-206 standardization process. The proposal was eventually abandoned
36  * but this class accidentally remained in the source tree, and made its
37  * way into the final version.
38  * </p><p>
39  * This class does not participate in any JAXP 1.3 or JAXP 1.4 processing.
40  * It must not be used by users or JAXP implementations.
41  * </p>
42  *
43  * @author <a href="Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
44  * @since 1.5
45  */
46  public abstract class SchemaFactoryLoader {
47
48      /**
49       * A do-nothing constructor.
50       */
51      protected SchemaFactoryLoader() {
52      }
53
54      /**
55       * Creates a new {@link SchemaFactory} object for the specified
56       * schema language.
57       *
58       * @param schemaLanguage
59       *     See <a href="SchemaFactory.html#schemaLanguage">
60       *     the list of available schema languages</a>.
61       *
62       * @throws NullPointerException
63       *     If the <tt>schemaLanguage</tt> parameter is null.
64       *
65       * @return <code>null</code> if the callee fails to create one.
66       */
67      public abstract SchemaFactory newFactory(String schemaLanguage);
68  }

```

## 26.6 SecuritySupport.java

Secuencia 256: javax/xml/validation/SecuritySupport.java

```

1  /*
2  * Copyright (c) 2005, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```

13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.validation;
27
28 import java.io.IOException;
29 import java.net.URL;
30 import java.security.*;
31 import java.net.*;
32 import java.io.*;
33 import java.util.*;
34
35 /**
36  * This class is duplicated for each JAXP subpackage so keep it in sync.
37  * It is package private and therefore is not exposed as part of the JAXP
38  * API.
39  *
40  * Security related methods that only work on J2SE 1.2 and newer.
41  */
42 class SecuritySupport {
43
44
45     ClassLoader getContextClassLoader() {
46         return (ClassLoader)
47             AccessController.doPrivileged(new PrivilegedAction() {
48                 public Object run() {
49                     ClassLoader cl = null;
50                     //try {
51                         cl = Thread.currentThread().getContextClassLoader();
52                     //} catch (SecurityException ex) { }
53                     if (cl == null)
54                         cl = ClassLoader.getSystemClassLoader();
55                     return cl;
56                 }
57             });
58     }
59
60     String getSystemProperty(final String propName) {
61         return (String)
62             AccessController.doPrivileged(new PrivilegedAction() {
63                 public Object run() {
64                     return System.getProperty(propName);
65                 }
66             });
67     }
68 }

```

```
66         });
67     }
68
69     FileInputStream getFileInputStream(final File file)
70         throws FileNotFoundException
71     {
72         try {
73             return (FileInputStream)
74                 AccessController.doPrivileged(new PrivilegedExceptionAction() {
75                     public Object run() throws FileNotFoundException {
76                         return new FileInputStream(file);
77                     }
78                 });
79         } catch (PrivilegedActionException e) {
80             throw (FileNotFoundException)e.getException();
81         }
82     }
83
84     InputStream getURLInputStream(final URL url)
85         throws IOException
86     {
87         try {
88             return (InputStream)
89                 AccessController.doPrivileged(new PrivilegedExceptionAction() {
90                     public Object run() throws IOException {
91                         return url.openStream();
92                     }
93                 });
94         } catch (PrivilegedActionException e) {
95             throw (IOException)e.getException();
96         }
97     }
98
99     URL getResourceAsURL(final ClassLoader cl,
100                        final String name)
101     {
102         return (URL)
103             AccessController.doPrivileged(new PrivilegedAction() {
104                 public Object run() {
105                     URL url;
106                     if (cl == null) {
107                         url = Object.class.getResource(name);
108                     } else {
109                         url = cl.getResource(name);
110                     }
111                     return url;
112                 }
113             });
114     }
115
116     Enumeration getResources(final ClassLoader cl,
117                        final String name) throws IOException
118     {
```

```

119     try{
120     return (Enumeration)
121         AccessController.doPrivileged(new PrivilegedExceptionAction() {
122             public Object run() throws IOException{
123                 Enumeration enumeration;
124                 if (cl == null) {
125                     enumeration = ClassLoader.getSystemResources(name);
126                 } else {
127                     enumeration = cl.getResources(name);
128                 }
129                 return enumeration;
130             }
131         });
132     }catch(PrivilegedActionException e){
133         throw (IOException)e.getException();
134     }
135 }
136
137 InputStream getResourceAsStream(final ClassLoader cl,
138                                 final String name)
139 {
140     return (InputStream)
141         AccessController.doPrivileged(new PrivilegedAction() {
142             public Object run() {
143                 InputStream ris;
144                 if (cl == null) {
145                     ris = Object.class.getResourceAsStream(name);
146                 } else {
147                     ris = cl.getResourceAsStream(name);
148                 }
149                 return ris;
150             }
151         });
152 }
153
154 boolean doesFileExist(final File f) {
155     return ((Boolean)
156         AccessController.doPrivileged(new PrivilegedAction() {
157             public Object run() {
158                 return new Boolean(f.exists());
159             }
160         })).booleanValue();
161 }
162
163 }

```

## 26.7 TypeInfoProvider.java

Secuencia 257: javax/xml/validation/TypeInfoProvider.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *

```

```
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.validation;
27
28 import org.w3c.dom.TypeInfo;
29
30 /**
31  * This class provides access to the type information determined
32  * by {@link ValidatorHandler}.
33  *
34  * <p>
35  * Some schema languages, such as W3C XML Schema, encourages a validator
36  * to report the "type" it assigns to each attribute/element.
37  * Those applications who wish to access this type information can invoke
38  * methods defined on this "interface" to access such type information.
39  *
40  * <p>
41  * Implementation of this "interface" can be obtained through the
42  * {@link ValidatorHandler#getTypeInfoProvider()} method.
43  *
44  * @author <a href="mailto:Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
45  * @see org.w3c.dom.TypeInfo
46  * @since 1.5
47  */
48 public abstract class TypeInfoProvider {
49
50     /**
51      * Constructor for the derived class.
52      *
53      * <p>
54      * The constructor does nothing.
55      */
56     protected TypeInfoProvider() {
57     }
```



```

58
59 /**
60  * <p>Returns the immutable {@link TypeInfo} object for the current
61  * element.</p>
62  *
63  * <p>The method may only be called by the startElement event
64  * or the endElement event
65  * of the {@link org.xml.sax.ContentHandler} that the application sets to
66  * the {@link ValidatorHandler}.</p>
67  *
68  * <p>When W3C XML Schema validation is being performed, in the
69  * case where an element has a union type, the {@link TypeInfo}
70  * returned by a call to <code>getElementTypeInfo()</code> from the
71  * startElement
72  * event will be the union type. The <code>TypeInfo</code>
73  * returned by a call
74  * from the endElement event will be the actual member type used
75  * to validate the element.</p>
76  *
77  * @throws IllegalStateException
78  *     If this method is called from other {@link org.xml.sax.ContentHandler}
79  *     methods.
80  * @return
81  *     An immutable {@link TypeInfo} object that represents the
82  *     type of the current element.
83  *     Note that the caller can keep references to the obtained
84  *     {@link TypeInfo} longer than the callback scope.
85  *
86  *     Otherwise, this method returns
87  *     null if the validator is unable to
88  *     determine the type of the current element for some reason
89  *     (for example, if the validator is recovering from
90  *     an earlier error.)
91  *
92  */
93 public abstract TypeInfo getElementTypeInfo();
94
95 /**
96  * Returns the immutable {@link TypeInfo} object for the specified
97  * attribute of the current element.
98  *
99  * <p>
100  * The method may only be called by the startElement event of
101  * the {@link org.xml.sax.ContentHandler} that the application sets to the
102  * {@link ValidatorHandler}.</p>
103  *
104  * @param index
105  *     The index of the attribute. The same index for
106  *     the {@link org.xml.sax.Attributes} object passed to the
107  *     <code>startElement</code> callback.
108  *
109  * @throws IndexOutOfBoundsException
110  *     If the index is invalid.

```

```

111     * @throws IllegalStateException
112     *     If this method is called from other {@link org.xml.sax.ContentHandler}
113     *     methods.
114     *
115     * @return
116     *     An immutable {@link TypeInfo} object that represents the
117     *     type of the specified attribute.
118     *     Note that the caller can keep references to the obtained
119     *     {@link TypeInfo} longer than the callback scope.
120     *
121     *     Otherwise, this method returns
122     *     null if the validator is unable to
123     *     determine the type.
124     */
125     public abstract TypeInfo getAttributeTypeInfo(int index);
126
127     /**
128     * Returns true if the specified attribute is determined
129     * to be ID.
130     *
131     * <p>
132     * Exactly how an attribute is "determined to be ID" is up to the
133     * schema language. In case of W3C XML Schema, this means
134     * that the actual type of the attribute is the built-in ID type
135     * or its derived type.
136     *
137     * <p>
138     * A {@link javax.xml.parsers.DocumentBuilder} uses this information
139     * to properly implement {@link org.w3c.dom.Attr#isId()}.
140     *
141     * <p>
142     * The method may only be called by the startElement event of
143     * the {@link org.xml.sax.ContentHandler} that the application sets to the
144     * {@link ValidatorHandler}.
145     *
146     * @param index
147     *     The index of the attribute. The same index for
148     *     the {@link org.xml.sax.Attributes} object passed to the
149     *     startElement callback.
150     *
151     * @throws IndexOutOfBoundsException
152     *     If the index is invalid.
153     * @throws IllegalStateException
154     *     If this method is called from other {@link org.xml.sax.ContentHandler}
155     *     methods.
156     *
157     * @return true
158     *     if the type of the specified attribute is ID.
159     */
160     public abstract boolean isIdAttribute(int index);
161
162     /**
163     * Returns false if the attribute was added by the validator.

```

```

164      *
165      * <p>
166      * This method provides information necessary for
167      * a {@link javax.xml.parsers.DocumentBuilder} to determine what
168      * the DOM tree should return from the {@link org.w3c.dom.Attr#getSpecified()} method.
169      *
170      * <p>
171      * The method may only be called by the startElement event of
172      * the {@link org.xml.sax.ContentHandler} that the application sets to the
173      * {@link ValidatorHandler}.
174      *
175      * <p>
176      * A general guideline for validators is to return true if
177      * the attribute was originally present in the pipeline, and
178      * false if it was added by the validator.
179      *
180      * @param index
181      *     The index of the attribute. The same index for
182      *     the {@link org.xml.sax.Attributes} object passed to the
183      *     <code>startElement</code> callback.
184      *
185      * @throws IndexOutOfBoundsException
186      *     If the index is invalid.
187      * @throws IllegalStateException
188      *     If this method is called from other {@link org.xml.sax.ContentHandler}
189      *     methods.
190      *
191      * @return
192      *     <code>true</code> if the attribute was present before the validator
193      *     processes input. <code>false</code> if the attribute was added
194      *     by the validator.
195      */
196     public abstract boolean isSpecified(int index);
197 }

```

## 26.8 Validator.java

Secuencia 258: javax/xml/validation/Validator.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.validation;
27
28  import java.io.IOException;
29
30  import javax.xml.transform.Result;
31  import javax.xml.transform.Source;
32
33  import org.w3c.dom.ls.LSResourceResolver;
34  import org.xml.sax.ErrorHandler;
35  import org.xml.sax.SAXException;
36  import org.xml.sax.SAXNotRecognizedException;
37  import org.xml.sax.SAXNotSupportedException;
38
39  /**
40   * <p>A processor that checks an XML document against {@link Schema}.</p>
41   *
42   * <p>
43   * A validator object is not thread-safe and not reentrant.
44   * In other words, it is the application's responsibility to make
45   * sure that one {@link Validator} object is not used from
46   * more than one thread at any given time, and while the <code>validate</code>
47   * method is invoked, applications may not recursively call
48   * the <code>validate</code> method.
49   * <p>
50   *
51   *
52   * @author <a href="mailto:Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
53   * @since 1.5
54   */
55  public abstract class Validator {
56
57      /**
58       * Constructor for derived classes.
59       *
60       * <p>The constructor does nothing.</p>
61       *
62       * <p>Derived classes must create {@link Validator} objects that have
63       * <code>null</code> {@link ErrorHandler} and
64       * <code>null</code> {@link LSResourceResolver}.
65       * </p>
66       */
67      protected Validator() {
68      }
```

```

69
70 /**
71  * <p>Reset this <code>Validator</code> to its original configuration.</p>
72  *
73  * <p><code>Validator</code> is reset to the same state as when it was created with
74  * {@link Schema#newValidator()}.
75  * <code>reset()</code> is designed to allow the reuse of existing <code>Validator</code>s
76  * thus saving resources associated with the creation of new <code>Validator</code>s.</p>
77  *
78  * <p>The reset <code>Validator</code> is not guaranteed to have the same {@link
79  *   LSResourceResolver} or {@link ErrorHandler}
80  * <code>Object</code>s, e.g. {@link Object#equals(Object obj)}. It is guaranteed to have
81  *   a functionally equal
82  * <code>LSResourceResolver</code> and <code>ErrorHandler</code>.</p>
83  */
84 public abstract void reset();
85
86 /**
87  * Validates the specified input.
88  *
89  * <p>This is just a convenience method for
90  * {@link #validate(Source source, Result result)}
91  * with <code>result</code> of <code>null</code>.</p>
92  *
93  * @param source
94  *   XML to be validated. Must be an XML document or
95  *   XML element and must not be null. For backwards compatibility,
96  *   the results of attempting to validate anything other than
97  *   a document or element are implementation-dependent.
98  *   Implementations must either recognize and process the input
99  *   or throw an IllegalArgumentException.
100  *
101  * @throws IllegalArgumentException
102  *   If the <code>Source</code>
103  *   is an XML artifact that the implementation cannot
104  *   validate (for example, a processing instruction).
105  *
106  * @throws SAXException
107  *   If the {@link ErrorHandler} throws a {@link SAXException} or
108  *   if a fatal error is found and the {@link ErrorHandler} returns
109  *   normally.
110  *
111  * @throws IOException
112  *   If the validator is processing a
113  *   {@link javax.xml.transform.sax.SAXSource} and the
114  *   underlying {@link org.xml.sax.XMLReader} throws an
115  *   {@link IOException}.
116  *
117  * @throws NullPointerException If <code>source</code> is
118  *   <code>null</code>.
119  *
120  * @see #validate(Source source, Result result)

```

```

120     */
121     public void validate(Source source)
122         throws SAXException, IOException {
123
124         validate(source, null);
125     }
126
127     /**
128     * <p>Validates the specified input and send the augmented validation
129     * result to the specified output.</p>
130     *
131     * <p>This method places the following restrictions on the types of
132     * the {@link Source}/{@link Result} accepted.</p>
133     *
134     * <table border=1>
135     * <thead>
136     * <tr>
137     * <th colspan="5"><code>Source</code> / <code>Result</code> Accepted</th>
138     * </tr>
139     * <tr>
140     * <th></th>
141     * <th>{@link javax.xml.transform.stream.StreamSource}</th>
142     * <th>{@link javax.xml.transform.sax.SAXSource}</th>
143     * <th>{@link javax.xml.transform.dom.DOMSource}</th>
144     * <th>{@link javax.xml.transform.stax.StAXSource}</th>
145     * </tr>
146     * </thead>
147     * <tbody align="center">
148     * <tr>
149     * <td><code>null</code></td>
150     * <td>OK</td>
151     * <td>OK</td>
152     * <td>OK</td>
153     * <td>OK</td>
154     * </tr>
155     * <tr>
156     * <th>{@link javax.xml.transform.stream.StreamResult}</th>
157     * <td>OK</td>
158     * <td><code>IllegalArgumentException</code></td>
159     * <td><code>IllegalArgumentException</code></td>
160     * <td><code>IllegalArgumentException</code></td>
161     * </tr>
162     * <tr>
163     * <th>{@link javax.xml.transform.sax.SAXResult}</th>
164     * <td><code>IllegalArgumentException</code></td>
165     * <td>OK</td>
166     * <td><code>IllegalArgumentException</code></td>
167     * <td><code>IllegalArgumentException</code></td>
168     * </tr>
169     * <tr>
170     * <th>{@link javax.xml.transform.dom.DOMResult}</th>
171     * <td><code>IllegalArgumentException</code></td>
172     * <td><code>IllegalArgumentException</code></td>

```

```

173 * <td>OK</td>
174 * <td><code>IllegalArgumentException</code></td>
175 * </tr>
176 * <tr>
177 * <th>{@link javax.xml.transform.stax.StAXResult}</th>
178 * <td><code>IllegalArgumentException</code></td>
179 * <td><code>IllegalArgumentException</code></td>
180 * <td><code>IllegalArgumentException</code></td>
181 * <td>OK</td>
182 * </tr>
183 * </tbody>
184 * </table>
185 *
186 * <p>To validate one <code>Source</code> into another kind of
187 * <code>Result</code>, use the identity transformer (see
188 * {@link javax.xml.transform.TransformerFactory#newTransformer()}).</p>
189 *
190 * <p>Errors found during the validation is sent to the specified
191 * {@link ErrorHandler}.</p>
192 *
193 * <p>If a document is valid, or if a document contains some errors
194 * but none of them were fatal and the <code>ErrorHandler</code> didn't
195 * throw any exception, then the method returns normally.</p>
196 *
197 * @param source
198 *     XML to be validated. Must be an XML document or
199 *     XML element and must not be null. For backwards compatibility,
200 *     the results of attempting to validate anything other than
201 *     a document or element are implementation-dependent.
202 *     Implementations must either recognize and process the input
203 *     or throw an IllegalArgumentException.
204 *
205 * @param result
206 *     The <code>Result</code> object that receives (possibly augmented)
207 *     XML. This parameter can be null if the caller is not interested
208 *     in it.
209 *
210 *     Note that when a <code>DOMResult</code> is used,
211 *     a validator might just pass the same DOM node from
212 *     <code>DOMSource</code> to <code>DOMResult</code>
213 *     (in which case <code>source.getNode()==result.getNode()</code>),
214 *     it might copy the entire DOM tree, or it might alter the
215 *     node given by the source.
216 *
217 * @throws IllegalArgumentException
218 *     If the <code>Result</code> type doesn't match the
219 *     <code>Source</code> type or if the <code>Source</code>
220 *     is an XML artifact that the implementation cannot
221 *     validate (for example, a processing instruction).
222 * @throws SAXException
223 *     If the <code>ErrorHandler</code> throws a
224 *     <code>SAXException</code> or
225 *     if a fatal error is found and the <code>ErrorHandler</code> returns

```

```

226 *      normally.
227 * @throws IOException
228 *      If the validator is processing a
229 *      <code>SAXSource</code> and the
230 *      underlying {@link org.xml.sax.XMLReader} throws an
231 *      <code>IOException</code>.
232 * @throws NullPointerException
233 *      If the <code>source</code> parameter is <code>null</code>.
234 *
235 * @see #validate(Source source)
236 */
237 public abstract void validate(Source source, Result result)
238     throws SAXException, IOException;
239
240 /**
241 * Sets the {@link ErrorHandler} to receive errors encountered
242 * during the <code>validate</code> method invocation.
243 *
244 * <p>
245 * Error handler can be used to customize the error handling process
246 * during a validation. When an {@link ErrorHandler} is set,
247 * errors found during the validation will be first sent
248 * to the {@link ErrorHandler}.
249 *
250 * <p>
251 * The error handler can abort further validation immediately
252 * by throwing {@link SAXException} from the handler. Or for example
253 * it can print an error to the screen and try to continue the
254 * validation by returning normally from the {@link ErrorHandler}
255 *
256 * <p>
257 * If any {@link Throwable} is thrown from an {@link ErrorHandler},
258 * the caller of the <code>validate</code> method will be thrown
259 * the same {@link Throwable} object.
260 *
261 * <p>
262 * {@link Validator} is not allowed to
263 * throw {@link SAXException} without first reporting it to
264 * {@link ErrorHandler}.
265 *
266 * <p>
267 * When the {@link ErrorHandler} is null, the implementation will
268 * behave as if the following {@link ErrorHandler} is set:
269 * <pre>
270 * class DraconianErrorHandler implements {@link ErrorHandler} {
271 *     public void fatalError( {@link org.xml.sax.SAXParseException} e ) throws {@link
272 *         SAXException} {
273 *         throw e;
274 *     }
275 *     public void error( {@link org.xml.sax.SAXParseException} e ) throws {@link SAXException}
276 *     {
277 *         throw e;
278 *     }
279 * }

```



```

277     *      public void warning( {@link org.xml.sax.SAXParseException} e ) throws {@link
          SAXException} {
278     *          // noop
279     *      }
280     *  }
281     *  </pre>
282     *
283     *  <p>
284     *  When a new {@link Validator} object is created, initially
285     *  this field is set to null.
286     *
287     *  @param    errorHandler
288     *      A new error handler to be set. This parameter can be null.
289     */
290     public abstract void setErrorHandler(ErrorHandler errorHandler);
291
292     /**
293     *  Gets the current {@link ErrorHandler} set to this {@link Validator}.
294     *
295     *  @return
296     *      This method returns the object that was last set through
297     *      the {@link #setErrorHandler(ErrorHandler)} method, or null
298     *      if that method has never been called since this {@link Validator}
299     *      has created.
300     *
301     *  @see #setErrorHandler(ErrorHandler)
302     */
303     public abstract ErrorHandler getErrorHandler();
304
305     /**
306     *  Sets the {@link LSResourceResolver} to customize
307     *  resource resolution while in a validation episode.
308     *
309     *  <p>
310     *  {@link Validator} uses a {@link LSResourceResolver}
311     *  when it needs to locate external resources while a validation,
312     *  although exactly what constitutes "locating external resources" is
313     *  up to each schema language.
314     *
315     *  <p>
316     *  When the {@link LSResourceResolver} is null, the implementation will
317     *  behave as if the following {@link LSResourceResolver} is set:
318     *  <pre>
319     *  class DumbLSResourceResolver implements {@link LSResourceResolver} {
320     *      public {@link org.w3c.dom.ls.LSInput} resolveResource(
321     *          String publicId, String systemId, String baseURI) {
322     *
323     *          return null; // always return null
324     *      }
325     *  }
326     *  </pre>
327     *
328     *  <p>

```

```

329 * If a {@link LSResourceResolver} throws a {@link RuntimeException}
330 * (or instances of its derived classes),
331 * then the {@link Validator} will abort the parsing and
332 * the caller of the <code>validate</code> method will receive
333 * the same {@link RuntimeException}.
334 *
335 * <p>
336 * When a new {@link Validator} object is created, initially
337 * this field is set to null.
338 *
339 * @param resourceResolver
340 *     A new resource resolver to be set. This parameter can be null.
341 */
342 public abstract void setResourceResolver(LSResourceResolver resourceResolver);
343
344 /**
345 * Gets the current {@link LSResourceResolver} set to this {@link Validator}.
346 *
347 * @return
348 *     This method returns the object that was last set through
349 *     the {@link #setResourceResolver(LSResourceResolver)} method, or null
350 *     if that method has never been called since this {@link Validator}
351 *     has created.
352 *
353 * @see #setErrorHandler(ErrorHandler)
354 */
355 public abstract LSResourceResolver getResourceResolver();
356
357
358
359 /**
360 * Look up the value of a feature flag.
361 *
362 * <p>The feature name is any fully-qualified URI. It is
363 * possible for a {@link Validator} to recognize a feature name but
364 * temporarily be unable to return its value.
365 * Some feature values may be available only in specific
366 * contexts, such as before, during, or after a validation.
367 *
368 * <p>Implementors are free (and encouraged) to invent their own features,
369 * using names built on their own URIs.</p>
370 *
371 * @param name The feature name, which is a non-null fully-qualified URI.
372 *
373 * @return The current value of the feature (true or false).
374 *
375 * @throws SAXNotRecognizedException If the feature
376 *     value can't be assigned or retrieved.
377 * @throws SAXNotSupportedException When the
378 *     {@link Validator} recognizes the feature name but
379 *     cannot determine its value at this time.
380 * @throws NullPointerException
381 *     When the name parameter is null.

```

```

382     *
383     * @see #setFeature(String, boolean)
384     */
385     public boolean getFeature(String name)
386         throws SAXNotRecognizedException, SAXNotSupportedException {
387
388         if (name == null) {
389             throw new NullPointerException("the name parameter is null");
390         }
391
392         throw new SAXNotRecognizedException(name);
393     }
394
395     /**
396     * Set the value of a feature flag.
397     *
398     * <p>
399     * Feature can be used to control the way a {@link Validator}
400     * parses schemas, although {@link Validator}s are not required
401     * to recognize any specific feature names.</p>
402     *
403     * <p>The feature name is any fully-qualified URI. It is
404     * possible for a {@link Validator} to expose a feature value but
405     * to be unable to change the current value.
406     * Some feature values may be immutable or mutable only
407     * in specific contexts, such as before, during, or after
408     * a validation.</p>
409     *
410     * @param name The feature name, which is a non-null fully-qualified URI.
411     * @param value The requested value of the feature (true or false).
412     *
413     * @throws SAXNotRecognizedException If the feature
414     *     value can't be assigned or retrieved.
415     * @throws SAXNotSupportedException When the
416     *     {@link Validator} recognizes the feature name but
417     *     cannot set the requested value.
418     * @throws NullPointerException
419     *     When the name parameter is null.
420     *
421     * @see #getFeature(String)
422     */
423     public void setFeature(String name, boolean value)
424         throws SAXNotRecognizedException, SAXNotSupportedException {
425
426         if (name == null) {
427             throw new NullPointerException("the name parameter is null");
428         }
429
430         throw new SAXNotRecognizedException(name);
431     }
432
433     /**
434     * Set the value of a property.

```

```

435 *
436 * <p>The property name is any fully-qualified URI. It is
437 * possible for a {@link Validator} to recognize a property name but
438 * to be unable to change the current value.
439 * Some property values may be immutable or mutable only
440 * in specific contexts, such as before, during, or after
441 * a validation.</p>
442 *
443 * <p>
444 * All implementations that implement JAXP 1.5 or newer are required to
445 * support the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD} and
446 * {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} properties.
447 * </p>
448 * <ul>
449 *   <li>
450 *     <p>Access to external DTDs in source or Schema file is restricted to
451 *     the protocols specified by the {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_DTD}
452 *     property. If access is denied during validation due to the restriction
453 *     of this property, {@link org.xml.sax.SAXException} will be thrown by the
454 *     {@link #validate(Source)} method.</p>
455 *
456 *     <p>Access to external reference set by the schemaLocation attribute is
457 *     restricted to the protocols specified by the
458 *     {@link javax.xml.XMLConstants#ACCESS_EXTERNAL_SCHEMA} property.
459 *     If access is denied during validation due to the restriction of this property,
460 *     {@link org.xml.sax.SAXException} will be thrown by the
461 *     {@link #validate(Source)} method.</p>
462 *   </li>
463 * </ul>
464 *
465 * @param name The property name, which is a non-null fully-qualified URI.
466 * @param object The requested value for the property.
467 *
468 * @throws SAXNotRecognizedException If the property
469 *   value can't be assigned or retrieved.
470 * @throws SAXNotSupportedException When the
471 *   {@link Validator} recognizes the property name but
472 *   cannot set the requested value.
473 * @throws NullPointerException
474 *   When the name parameter is null.
475 */
476 public void setProperty(String name, Object object)
477     throws SAXNotRecognizedException, SAXNotSupportedException {
478
479     if (name == null) {
480         throw new NullPointerException("the name parameter is null");
481     }
482
483     throw new SAXNotRecognizedException(name);
484 }
485
486 /**
487 * Look up the value of a property.

```

```

488      *
489      * <p>The property name is any fully-qualified URI. It is
490      * possible for a {@link Validator} to recognize a property name but
491      * temporarily be unable to return its value.
492      * Some property values may be available only in specific
493      * contexts, such as before, during, or after a validation.</p>
494      *
495      * <p>{@link Validator}s are not required to recognize any specific
496      * property names.</p>
497      *
498      * <p>Implementors are free (and encouraged) to invent their own properties,
499      * using names built on their own URIs.</p>
500      *
501      * @param name The property name, which is a non-null fully-qualified URI.
502      *
503      * @return The current value of the property.
504      *
505      * @throws SAXNotRecognizedException If the property
506      * value can't be assigned or retrieved.
507      * @throws SAXNotSupportedException When the
508      * XMLReader recognizes the property name but
509      * cannot determine its value at this time.
510      * @throws NullPointerException
511      * When the name parameter is null.
512      *
513      * @see #setProperty(String, Object)
514      */
515     public Object getProperty(String name)
516         throws SAXNotRecognizedException, SAXNotSupportedException {
517
518         if (name == null) {
519             throw new NullPointerException("the name parameter is null");
520         }
521
522         throw new SAXNotRecognizedException(name);
523     }
524 }

```

## 26.9 ValidatorHandler.java

Secuencia 259: javax/xml/validation/ValidatorHandler.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.validation;
27
28 import org.w3c.dom.ls.LSResourceResolver;
29 import org.xml.sax.ContentHandler;
30 import org.xml.sax.ErrorHandler;
31 import org.xml.sax.SAXNotRecognizedException;
32 import org.xml.sax.SAXNotSupportedException;
33
34 /**
35  * Streaming validator that works on SAX stream.
36  *
37  * <p>
38  * A {@link ValidatorHandler} object is not thread-safe and not reentrant.
39  * In other words, it is the application's responsibility to make
40  * sure that one {@link ValidatorHandler} object is not used from
41  * more than one thread at any given time.
42  *
43  * <p>
44  * {@link ValidatorHandler} checks if the SAX events follow
45  * the set of constraints described in the associated {@link Schema},
46  * and additionally it may modify the SAX events (for example
47  * by adding default values, etc.)
48  *
49  * <p>
50  * {@link ValidatorHandler} extends from {@link ContentHandler},
51  * but it refines the underlying {@link ContentHandler} in
52  * the following way:
53  * <ol>
54  * <li>startElement/endElement events must receive non-null String
55  *     for <code>uri</code>, <code>localName</code>, and <code>qname</code>,
56  *     even though SAX allows some of them to be null.
57  *     Similarly, the user-specified {@link ContentHandler} will receive non-null
58  *     Strings for all three parameters.
59  *
60  * <li>Applications must ensure that {@link ValidatorHandler}'s
61  *     {@link ContentHandler#startPrefixMapping(String,String)} and
62  *     {@link ContentHandler#endPrefixMapping(String)} are invoked
63  *     properly. Similarly, the user-specified {@link ContentHandler}
64  *     will receive startPrefixMapping/endPrefixMapping events.
65  *     If the {@link ValidatorHandler} introduces additional namespace
```

```

66 *     bindings, the user-specified {@link ContentHandler} will receive
67 *     additional startPrefixMapping/endPrefixMapping events.
68 *
69 * <li>{@link org.xml.sax.Attributes} for the
70 *     {@link ContentHandler#startElement(String,String,String,Attributes)} method
71 *     may or may not include xmlns* attributes.
72 * </ol>
73 *
74 * <p>
75 * A {@link ValidatorHandler} is automatically reset every time
76 * the startDocument method is invoked.
77 *
78 * <h2>Recognized Properties and Features</h2>
79 * <p>
80 * This spec defines the following feature that must be recognized
81 * by all {@link ValidatorHandler} implementations.
82 *
83 * <h3><code>http://xml.org/sax/features/namespace-prefixes</code></h3>
84 * <p>
85 * This feature controls how a {@link ValidatorHandler} introduces
86 * namespace bindings that were not present in the original SAX event
87 * stream.
88 * When this feature is set to true, it must make
89 * sure that the user's {@link ContentHandler} will see
90 * the corresponding <code>xmlns*</code> attribute in
91 * the {@link org.xml.sax.Attributes} object of the
92 * {@link ContentHandler#startElement(String,String,String,Attributes)}
93 * callback. Otherwise, <code>xmlns*</code> attributes must not be
94 * added to {@link org.xml.sax.Attributes} that's passed to the
95 * user-specified {@link ContentHandler}.
96 * <p>
97 * (Note that regardless of this switch, namespace bindings are
98 * always notified to applications through
99 * {@link ContentHandler#startPrefixMapping(String,String)} and
100 * {@link ContentHandler#endPrefixMapping(String)} methods of the
101 * {@link ContentHandler} specified by the user.)
102 *
103 * <p>
104 * Note that this feature does <em>NOT</em> affect the way
105 * a {@link ValidatorHandler} receives SAX events. It merely
106 * changes the way it augments SAX events.
107 *
108 * <p>This feature is set to <code>>false</code> by default.</p>
109 *
110 * @author <a href="mailto:Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
111 * @since 1.5
112 */
113 public abstract class ValidatorHandler implements ContentHandler {
114
115     /**
116     * <p>Constructor for derived classes.</p>
117     *
118     * <p>The constructor does nothing.</p>

```

```

119      *
120      * <p>Derived classes must create {@link ValidatorHandler} objects that have
121      * <code>null</code> {@link ErrorHandler} and
122      * <code>null</code> {@link LSResourceResolver}</p>
123      */
124      protected ValidatorHandler() {
125      }
126
127      /**
128       * Sets the {@link ContentHandler} which receives
129       * the augmented validation result.
130       *
131       * <p>
132       * When a {@link ContentHandler} is specified, a
133       * {@link ValidatorHandler} will work as a filter
134       * and basically copy the incoming events to the
135       * specified {@link ContentHandler}.
136       *
137       * <p>
138       * In doing so, a {@link ValidatorHandler} may modify
139       * the events, for example by adding defaulted attributes.
140       *
141       * <p>
142       * A {@link ValidatorHandler} may buffer events to certain
143       * extent, but to allow {@link ValidatorHandler} to be used
144       * by a parser, the following requirement has to be met.
145       *
146       * <ol>
147       * <li>When
148       *     {@link ContentHandler#startElement(String, String, String, Attributes)},
149       *     {@link ContentHandler#endElement(String, String, String)},
150       *     {@link ContentHandler#startDocument()}, or
151       *     {@link ContentHandler#endDocument()}
152       *     are invoked on a {@link ValidatorHandler},
153       *     the same method on the user-specified {@link ContentHandler}
154       *     must be invoked for the same event before the callback
155       *     returns.
156       * <li>{@link ValidatorHandler} may not introduce new elements that
157       *     were not present in the input.
158       *
159       * <li>{@link ValidatorHandler} may not remove attributes that were
160       *     present in the input.
161       * </ol>
162       *
163       * <p>
164       * When a callback method on the specified {@link ContentHandler}
165       * throws an exception, the same exception object must be thrown
166       * from the {@link ValidatorHandler}. The {@link ErrorHandler}
167       * should not be notified of such an exception.
168       *
169       * <p>
170       * This method can be called even during a middle of a validation.
171       *

```



```

172     * @param receiver
173     *     A {@link ContentHandler} or a null value.
174     */
175     public abstract void setContentHandler(ContentHandler receiver);
176
177     /**
178     * Gets the {@link ContentHandler} which receives the
179     * augmented validation result.
180     *
181     * @return
182     *     This method returns the object that was last set through
183     *     the {@link #getContentHandler()} method, or null
184     *     if that method has never been called since this {@link ValidatorHandler}
185     *     has created.
186     *
187     * @see #setContentHandler(ContentHandler)
188     */
189     public abstract ContentHandler getContentHandler();
190
191     /**
192     * Sets the {@link ErrorHandler} to receive errors encountered
193     * during the validation.
194     *
195     * <p>
196     * Error handler can be used to customize the error handling process
197     * during a validation. When an {@link ErrorHandler} is set,
198     * errors found during the validation will be first sent
199     * to the {@link ErrorHandler}.
200     *
201     * <p>
202     * The error handler can abort further validation immediately
203     * by throwing {@link org.xml.sax.SAXException} from the handler. Or for example
204     * it can print an error to the screen and try to continue the
205     * validation by returning normally from the {@link ErrorHandler}
206     *
207     * <p>
208     * If any {@link Throwable} is thrown from an {@link ErrorHandler},
209     * the same {@link Throwable} object will be thrown toward the
210     * root of the call stack.
211     *
212     * <p>
213     * {@link ValidatorHandler} is not allowed to
214     * throw {@link org.xml.sax.SAXException} without first reporting it to
215     * {@link ErrorHandler}.
216     *
217     * <p>
218     * When the {@link ErrorHandler} is null, the implementation will
219     * behave as if the following {@link ErrorHandler} is set:
220     * <pre>
221     * class DraconianErrorHandler implements {@link ErrorHandler} {
222     *     public void fatalError( {@link org.xml.sax.SAXParseException} e ) throws {@link org.xml.
223     *         sax.SAXException} {
224     *         throw e;

```

```

224     *     }
225     *     public void error( {@link org.xml.sax.SAXParseException} e ) throws {@link org.xml.sax.
        SAXException} {
226     *         throw e;
227     *     }
228     *     public void warning( {@link org.xml.sax.SAXParseException} e ) throws {@link org.xml.sax
        .SAXException} {
229     *         // noop
230     *     }
231     * }
232     * </pre>
233     *
234     * <p>
235     * When a new {@link ValidatorHandler} object is created, initially
236     * this field is set to null.
237     *
238     * @param errorHandler
239     *     A new error handler to be set. This parameter can be null.
240     */
241     public abstract void setErrorHandler(ErrorHandler errorHandler);
242
243     /**
244     * Gets the current {@link ErrorHandler} set to this {@link ValidatorHandler}.
245     *
246     * @return
247     *     This method returns the object that was last set through
248     *     the {@link #setErrorHandler(ErrorHandler)} method, or null
249     *     if that method has never been called since this {@link ValidatorHandler}
250     *     has created.
251     *
252     * @see #setErrorHandler(ErrorHandler)
253     */
254     public abstract ErrorHandler getErrorHandler();
255
256     /**
257     * Sets the {@link LSResourceResolver} to customize
258     * resource resolution while in a validation episode.
259     *
260     * <p>
261     * {@link ValidatorHandler} uses a {@link LSResourceResolver}
262     * when it needs to locate external resources while a validation,
263     * although exactly what constitutes "locating external resources" is
264     * up to each schema language.
265     *
266     * <p>
267     * When the {@link LSResourceResolver} is null, the implementation will
268     * behave as if the following {@link LSResourceResolver} is set:
269     * <pre>
270     * class DumbLSResourceResolver implements {@link LSResourceResolver} {
271     *     public {@link org.w3c.dom.ls.LSInput} resolveResource(
272     *         String publicId, String systemId, String baseURI) {
273     *
274     *         return null; // always return null

```

```

275     *     }
276     * }
277     * </pre>
278     *
279     * <p>
280     * If a {@link LSResourceResolver} throws a {@link RuntimeException}
281     * (or instances of its derived classes),
282     * then the {@link ValidatorHandler} will abort the parsing and
283     * the caller of the <code>validate</code> method will receive
284     * the same {@link RuntimeException}.
285     *
286     * <p>
287     * When a new {@link ValidatorHandler} object is created, initially
288     * this field is set to null.
289     *
290     * @param   resourceResolver
291     *         A new resource resolver to be set. This parameter can be null.
292     */
293     public abstract void setResourceResolver(LSResourceResolver resourceResolver);
294
295     /**
296     * Gets the current {@link LSResourceResolver} set to this {@link ValidatorHandler}.
297     *
298     * @return
299     *         This method returns the object that was last set through
300     *         the {@link #setResourceResolver(LSResourceResolver)} method, or null
301     *         if that method has never been called since this {@link ValidatorHandler}
302     *         has created.
303     *
304     * @see #setErrorHandler(ErrorHandler)
305     */
306     public abstract LSResourceResolver getResourceResolver();
307
308     /**
309     * Obtains the {@link TypeInfoProvider} implementation of this
310     * {@link ValidatorHandler}.
311     *
312     * <p>
313     * The obtained {@link TypeInfoProvider} can be queried during a parse
314     * to access the type information determined by the validator.
315     *
316     * <p>
317     * Some schema languages do not define the notion of type,
318     * for those languages, this method may not be supported.
319     * However, to be compliant with this specification, implementations
320     * for W3C XML Schema 1.0 must support this operation.
321     *
322     * @return
323     *         null if the validator / schema language does not support
324     *         the notion of {@link org.w3c.dom.TypeInfo}.
325     *         Otherwise a non-null valid {@link TypeInfoProvider}.
326     */
327     public abstract TypeInfoProvider getTypeInfoProvider();

```

```

328
329
330 /**
331  * Look up the value of a feature flag.
332  *
333  * <p>The feature name is any fully-qualified URI. It is
334  * possible for a {@link ValidatorHandler} to recognize a feature name but
335  * temporarily be unable to return its value.
336  * Some feature values may be available only in specific
337  * contexts, such as before, during, or after a validation.
338  *
339  * <p>Implementors are free (and encouraged) to invent their own features,
340  * using names built on their own URIs.</p>
341  *
342  * @param name The feature name, which is a non-null fully-qualified URI.
343  *
344  * @return The current value of the feature (true or false).
345  *
346  * @throws SAXNotRecognizedException If the feature
347  * value can't be assigned or retrieved.
348  * @throws SAXNotSupportedException When the
349  * {@link ValidatorHandler} recognizes the feature name but
350  * cannot determine its value at this time.
351  * @throws NullPointerException When <code>name</code> is <code>null</code>.
352  *
353  * @see #setFeature(String, boolean)
354  */
355 public boolean getFeature(String name)
356     throws SAXNotRecognizedException, SAXNotSupportedException {
357
358     if (name == null) {
359         throw new NullPointerException();
360     }
361
362     throw new SAXNotRecognizedException(name);
363 }
364
365 /**
366  * <p>Set a feature for this <code>ValidatorHandler</code>.</p>
367  *
368  * <p>Feature can be used to control the way a
369  * {@link ValidatorHandler} parses schemas. The feature name is
370  * any fully-qualified URI. It is possible for a
371  * {@link SchemaFactory} to
372  * expose a feature value but to be unable to change the current
373  * value. Some feature values may be immutable or mutable only in
374  * specific contexts, such as before, during, or after a
375  * validation.</p>
376  *
377  * <p>All implementations are required to support the {@link javax.xml.XMLConstants#
378  *     FEATURE_SECURE_PROCESSING} feature.
379  * When the feature is:</p>
380  * <ul>

```

```

380 * <li>
381 *   <code>true</code>: the implementation will limit XML processing to conform to
      implementation limits.
382 *   Examples include entity expansion limits and XML Schema constructs that would consume
      large amounts of resources.
383 *   If XML processing is limited for security reasons, it will be reported via a call to the
      registered
384 *   {@link ErrorHandler#fatalError(SAXParseException exception)}.
385 *   See {@link #setErrorHandler(ErrorHandler errorHandler)}.
386 * </li>
387 * <li>
388 *   <code>false</code>: the implementation will processing XML according to the XML
      specifications without
389 *   regard to possible implementation limits.
390 * </li>
391 * </ul>
392 *
393 * @param name The feature name, which is a non-null fully-qualified URI.
394 * @param value The requested value of the feature (true or false).
395 *
396 * @throws SAXNotRecognizedException If the feature
397 *   value can't be assigned or retrieved.
398 * @throws SAXNotSupportedException When the
399 *   {@link ValidatorHandler} recognizes the feature name but
400 *   cannot set the requested value.
401 * @throws NullPointerException When <code>name</code> is <code>null</code>.
402 *
403 * @see #getFeature(String)
404 */
405 public void setFeature(String name, boolean value)
406     throws SAXNotRecognizedException, SAXNotSupportedException {
407
408     if (name == null) {
409         throw new NullPointerException();
410     }
411
412     throw new SAXNotRecognizedException(name);
413 }
414
415 /**
416 * Set the value of a property.
417 *
418 * <p>The property name is any fully-qualified URI. It is
419 * possible for a {@link ValidatorHandler} to recognize a property name but
420 * to be unable to change the current value.
421 * Some property values may be immutable or mutable only
422 * in specific contexts, such as before, during, or after
423 * a validation.</p>
424 *
425 * <p>{@link ValidatorHandler}s are not required to recognize setting
426 * any specific property names.</p>
427 *
428 * @param name The property name, which is a non-null fully-qualified URI.

```

```

429     * @param object The requested value for the property.
430     *
431     * @throws SAXNotRecognizedException If the property
432     *     value can't be assigned or retrieved.
433     * @throws SAXNotSupportedException When the
434     *     {@link ValidatorHandler} recognizes the property name but
435     *     cannot set the requested value.
436     * @throws NullPointerException When <code>name</code> is <code>null</code>.
437     */
438     public void setProperty(String name, Object object)
439         throws SAXNotRecognizedException, SAXNotSupportedException {
440
441         if (name == null) {
442             throw new NullPointerException();
443         }
444
445         throw new SAXNotRecognizedException(name);
446     }
447
448     /**
449     * Look up the value of a property.
450     *
451     * <p>The property name is any fully-qualified URI. It is
452     * possible for a {@link ValidatorHandler} to recognize a property name but
453     * temporarily be unable to return its value.
454     * Some property values may be available only in specific
455     * contexts, such as before, during, or after a validation.</p>
456     *
457     * <p>{@link ValidatorHandler}s are not required to recognize any specific
458     * property names.</p>
459     *
460     * <p>Implementors are free (and encouraged) to invent their own properties,
461     * using names built on their own URIs.</p>
462     *
463     * @param name The property name, which is a non-null fully-qualified URI.
464     *
465     * @return The current value of the property.
466     *
467     * @throws SAXNotRecognizedException If the property
468     *     value can't be assigned or retrieved.
469     * @throws SAXNotSupportedException When the
470     *     XMLReader recognizes the property name but
471     *     cannot determine its value at this time.
472     * @throws NullPointerException When <code>name</code> is <code>null</code>.
473     *
474     * @see #setProperty(String, Object)
475     */
476     public Object getProperty(String name)
477         throws SAXNotRecognizedException, SAXNotSupportedException {
478
479         if (name == null) {
480             throw new NullPointerException();
481         }

```

```

482
483     throw new SAXNotRecognizedException(name);
484 }
485 }

```

## 27 Xml Ws (javax.xml.ws package)

### 27.1 Action.java

Secuencia 260: javax/xml/ws/Action.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.ElementType;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.RetentionPolicy;
32 import java.lang.annotation.Target;
33
34 /**
35 * The <code>Action</code> annotation allows explicit association of a
36 * WS-Addressing <code>Action</code> message addressing property with
37 * <code>input</code>, <code>output</code>, and
38 * <code>fault</code> messages of the mapped WSDL operation.
39 * <p>
40 * This annotation can be specified on each method of a service endpoint interface.
41 * For such a method, the mapped operation in the generated WSDL's
42 * <code>wsam:Action</code> attribute on the WSDL <code>input</code>,
43 * <code>output</code> and <code>fault</code> messages of the WSDL <code>operation</code>

```

```

44 * is based upon which attributes of the <code>Action</code> annotation have been specified.
45 * For the exact computation of <code>wsam:Action</code> values for the messages, refer
46 * to the algorithm in the JAX-WS specification.
47 * <p>
48 * <b>Example 1</b>: Specify explicit values for <code>Action</code> message addressing property
49 * for <code>input</code> and <code>output</code> messages.
50 *
51 * <pre>
52 * &#64;WebService(targetNamespace="http://example.com/numbers")
53 * public class AddNumbersImpl {
54 *     <b>&#64;Action(
55 *         input="http://example.com/inputAction",
56 *         output="http://example.com/outputAction")</b>
57 *     public int addNumbers(int number1, int number2) {
58 *         return number1 + number2;
59 *     }
60 * }
61 * </pre>
62 *
63 * The generated WSDL looks like:
64 * <pre>
65 * <definitions targetNamespace="http://example.com/numbers" ...>
66 *     ...
67 *     <portType name="AddNumbersPortType">
68 *         <operation name="AddNumbers">
69 *             <input message="tns:AddNumbersInput" name="foo"
70 *                 <b>wsam:Action="http://example.com/inputAction"</b>/>
71 *             <output message="tns:AddNumbersOutput" name="bar"
72 *                 <b>wsam:Action="http://example.com/outputAction"</b>/>
73 *         </operation>
74 *     </portType>
75 *     ...
76 * </definitions>
77 * </pre>
78 *
79 * <p>
80 * <b>Example 2</b>: Specify explicit value for <code>Action</code> message addressing property
81 * for only the <code>input</code> message. The <code>wsam:Action</code> values for the
82 * WSDL <code>output</code> message are computed using the algorithm in the JAX-WS specification.
83 *
84 * <pre>
85 * &#64;WebService(targetNamespace="http://example.com/numbers")
86 * public class AddNumbersImpl {
87 *     <b>&#64;Action(input="http://example.com/inputAction")</b>
88 *     public int addNumbers(int number1, int number2) {
89 *         return number1 + number2;
90 *     }
91 * }
92 * </pre>
93 *
94 * The generated WSDL looks like:
95 * <pre>
96 * <definitions targetNamespace="http://example.com/numbers" ...>

```



```

97 *      ...
98 *      <portType name="AddNumbersPortType">
99 *          <operation name="AddNumbers">
100 *              <input message="tns:AddNumbersInput" name="foo"
101 *                  <b>wsam:Action="http://example.com/inputAction"</b> />
102 *              <output message="tns:AddNumbersOutput" name="bar"
103 *                  <b>wsam:Action="http://example.com/numbers/AddNumbersPortType/AddNumbersResponse"</b>
104 *              </operation>
105 *          </portType>
106 *      ...
107 *  </definitions>
108 * </pre>
109 *
110 * It is legitimate to specify an explicit value for <code>Action</code> message addressing
111 * property for
112 * <code>output</code> message only. In this case, <code>wsam:Action</code> value for the
113 * WSDL <code>input</code> message is computed using the algorithm in the JAX-WS specification.
114 *
115 * <p>
116 * <b>Example 3</b>: See {@link FaultAction} annotation for an example of
117 * how to specify an explicit value for <code>Action</code> message addressing property for the
118 * <code>fault</code> message.
119 *
120 * @see FaultAction
121 *
122 * @since JAX-WS 2.1
123 */
124 @Documented
125 @Retention(RetentionPolicy.RUNTIME)
126 @Target(ElementType.METHOD)
127 public @interface Action {
128     /**
129      * Explicit value of the WS-Addressing <code>Action</code> message addressing property for the
130      * <code>input</code>
131      * message of the operation.
132      */
133     String input() default "";
134
135     /**
136      * Explicit value of the WS-Addressing <code>Action</code> message addressing property for the
137      * <code>output</code>
138      * message of the operation.
139      */
140     String output() default "";
141
142     /**
143      * Explicit value of the WS-Addressing <code>Action</code> message addressing property for the
144      * <code>fault</code>
145      * message(s) of the operation. Each exception that is mapped to a fault and requires an
146      * explicit WS-Addressing
147      * <code>Action</code> message addressing property, needs to be specified as a value in this

```

```

    property
144     * using {@link FaultAction} annotation.
145     */
146     FaultAction[] fault() default { };
147 }

```

## 27.2 AsyncHandler.java

Secuencia 261: javax/xml/ws/AsyncHandler.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 /** The <code>AsyncHandler</code> interface is implemented by
29  * clients that wish to receive callback notification of the completion of
30  * service endpoint operations invoked asynchronously.
31  *
32  * @since JAX-WS 2.0
33  */
34 public interface AsyncHandler<T> {
35
36     /** Called when the response to an asynchronous operation is available.
37     *
38     * @param res The response to the operation invocation.
39     *
40     */
41     void handleResponse(Response<T> res);
42 }

```

## 27.3 Binding.java

Secuencia 262: javax/xml/ws/Binding.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28
29 /** The <code>Binding</code> interface is the base interface
30 * for JAX-WS protocol bindings.
31 *
32 * @since JAX-WS 2.0
33 */
34 public interface Binding {
35
36     /**
37      * Gets a copy of the handler chain for a protocol binding instance.
38      * If the returned chain is modified a call to <code>setHandlerChain</code>
39      * is required to configure the binding instance with the new chain.
40      *
41      * @return java.util.List<Handler> Handler chain
42      */
43     public java.util.List<javax.xml.ws.handler.Handler> getHandlerChain();
44
45     /**
46      * Sets the handler chain for the protocol binding instance.
47      *
48      * @param chain A List of handler configuration entries
49      * @throws WebServiceException On an error in the configuration of
50      * the handler chain
51      * @throws java.lang.UnsupportedOperationException If this
52      * operation is not supported. This may be done to
```

```

53      *          avoid any overriding of a pre-configured handler
54      *          chain.
55      */
56      public void setHandlerChain(java.util.List<javax.xml.ws.handler.Handler> chain);
57
58      /**
59       * Get the URI for this binding instance.
60       *
61       * @return String The binding identifier for the port.
62       *         Never returns <code>null</code>
63       *
64       * @since JAX-WS 2.1
65       */
66      String getBindingID();
67  }

```

## 27.4 BindingProvider.java

Secuencia 263: javax/xml/ws/BindingProvider.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.util.Map;
29 import javax.xml.ws.wsaddressing.W3CEndpointReference;
30
31 /**
32 * The <code>BindingProvider</code> interface provides access to the
33 * protocol binding and associated context objects for request and
34 * response message processing.

```

```
35  *
36  * @since JAX-WS 2.0
37  *
38  * @see javax.xml.ws.Binding
39  **/
40  public interface BindingProvider {
41      /**
42       * Standard property: User name for authentication.
43       * <p>Type: <code>java.lang.String</code>
44       */
45      public static final String USERNAME_PROPERTY =
46          "javax.xml.ws.security.auth.username";
47
48      /**
49       * Standard property: Password for authentication.
50       * <p>Type: <code>java.lang.String</code>
51       */
52      public static final String PASSWORD_PROPERTY =
53          "javax.xml.ws.security.auth.password";
54
55      /**
56       * Standard property: Target service endpoint address. The
57       * URI scheme for the endpoint address specification MUST
58       * correspond to the protocol/transport binding for the
59       * binding in use.
60       * <p>Type: <code>java.lang.String</code>
61       */
62      public static final String ENDPOINT_ADDRESS_PROPERTY =
63          "javax.xml.ws.service.endpoint.address";
64
65      /**
66       * Standard property: This boolean property is used by a service
67       * client to indicate whether or not it wants to participate in
68       * a session with a service endpoint. If this property is set to
69       * <code>true</code>, the service client indicates that it wants the session
70       * to be maintained. If set to <code>false</code>, the session is not maintained.
71       * The default value for this property is <code>false</code>.
72       * <p>Type: <code>java.lang.Boolean</code>
73       */
74      public static final String SESSION_MAINTAIN_PROPERTY =
75          "javax.xml.ws.session.maintain";
76
77      /**
78       * Standard property for SOAPAction. This boolean property
79       * indicates whether or not the value of the
80       * <code>javax.xml.ws.soap.http.soapaction.uri</code> property
81       * is used for the value of the SOAPAction. The
82       * default value of this property is <code>false</code> indicating
83       * that the
84       * <code>javax.xml.ws.soap.http.soapaction.uri</code> property
85       * is not used for the value of the SOAPAction, however,
86       * if WS-Addressing is enabled, the default value is
87       * <code>true</code>.
```

```
88      *
89      * <p>Type: <code>java.lang.Boolean</code>
90      **/
91      public static final String SOAPACTION_USE_PROPERTY =
92          "javax.xml.ws.soap.http.soapaction.use";
93
94      /**
95       * Standard property for SOAPAction. Indicates the SOAPAction
96       * URI if the <code>javax.xml.ws.soap.http.soapaction.use</code>
97       * property is set to <code>true</code>. If WS-Addressing
98       * is enabled, this value will also be used for the value of the
99       * WS-Addressing Action header. If this property is not set,
100      * the default SOAPAction and WS-Addressing Action will be sent.
101      *
102      * <p>Type: <code>java.lang.String</code>
103      **/
104      public static final String SOAPACTION_URI_PROPERTY =
105          "javax.xml.ws.soap.http.soapaction.uri";
106
107      /**
108       * Get the context that is used to initialize the message context
109       * for request messages.
110       *
111       * Modifications to the request context do not affect the message context of
112       * either synchronous or asynchronous operations that have already been
113       * started.
114       *
115       * @return The context that is used in processing request messages.
116       **/
117      Map<String, Object> getRequestContext();
118
119      /**
120       * Get the context that resulted from processing a response message.
121       *
122       * The returned context is for the most recently completed synchronous
123       * operation. Subsequent synchronous operation invocations overwrite the
124       * response context. Asynchronous operations return their response context
125       * via the Response interface.
126       *
127       * @return The context that resulted from processing the latest
128       * response messages.
129       **/
130      Map<String, Object> getResponseContext();
131
132      /**
133       * Get the Binding for this binding provider.
134       *
135       * @return The Binding for this binding provider.
136       **/
137      Binding getBinding();
138
139
140
```

```

141 /**
142  * Returns the <code>EndpointReference</code> associated with
143  * this <code>BindingProvider</code> instance.
144  * <p>
145  * If the Binding for this <code>bindingProvider</code> is
146  * either SOAP1.1/HTTP or SOAP1.2/HTTP, then a
147  * <code>W3CEndpointReference</code> MUST be returned.
148  *
149  * @return EndpointReference of the target endpoint associated with this
150  * <code>BindingProvider</code> instance.
151  *
152  * @throws java.lang.UnsupportedOperationException If this
153  * <code>BindingProvider</code> uses the XML/HTTP binding.
154  *
155  * @see W3CEndpointReference
156  *
157  * @since JAX-WS 2.1
158  */
159 public EndpointReference getEndpointReference();
160
161
162 /**
163  * Returns the <code>EndpointReference</code> associated with
164  * this <code>BindingProvider</code> instance. The instance
165  * returned will be of type <code>clazz</code>.
166  *
167  * @param clazz Specifies the type of <code>EndpointReference</code>
168  * that MUST be returned.
169  *
170  * @return EndpointReference of the target endpoint associated with this
171  * <code>BindingProvider</code> instance. MUST be of type
172  * <code>clazz</code>.
173  *
174  * @throws WebServiceException If the Class <code>clazz</code>
175  * is not supported by this implementation.
176  * @throws java.lang.UnsupportedOperationException If this
177  * <code>BindingProvider</code> uses the XML/HTTP binding.
178  *
179  * @since JAX-WS 2.1
180  */
181 public <T extends EndpointReference> T getEndpointReference(Class<T> clazz);
182 }

```

## 27.5 BindingType.java

Secuencia 264: javax/xml/ws/BindingType.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *

```

```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.ElementType;
31 import java.lang.annotation.Retention;
32 import java.lang.annotation.RetentionPolicy;
33
34 /**
35  * The <code>BindingType</code> annotation is used to
36  * specify the binding to use for a web service
37  * endpoint implementation class.
38  * <p>
39  * This annotation may be overridden programmatically or via
40  * deployment descriptors, depending on the platform in use.
41  *
42  * @since JAX-WS 2.0
43  *
44  */
45 @Target(ElementType.TYPE)
46 @Retention(RetentionPolicy.RUNTIME)
47 @Documented
48 public @interface BindingType {
49     /**
50      * A binding identifier (a URI).
51      * If not specified, the default is the SOAP 1.1 / HTTP binding.
52      * <p>
53      * See the <code>SOAPBinding</code> and <code>HTTPBinding</code>
54      * for the definition of the standard binding identifiers.
55      *
56      * @see javax.xml.ws.Binding
57      * @see javax.xml.ws.soap.SOAPBinding#SOAP11HTTP_BINDING
58      * @see javax.xml.ws.soap.SOAPBinding#SOAP12HTTP_BINDING
59      * @see javax.xml.ws.http.HTTPBinding#HTTP_BINDING
60      */
61 }
```



```

61     String value() default "" ;
62 }

```

## 27.6 Dispatch.java

Secuencia 265: javax/xml/ws/Dispatch.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.util.concurrent.Future;
29
30 /** The <code>Dispatch</code> interface provides support
31 * for the dynamic invocation of a service endpoint operations. The
32 * <code>javax.xml.ws.Service</code>
33 * class acts as a factory for the creation of <code>Dispatch</code>
34 * instances.
35 *
36 * @since JAX-WS 2.0
37 */
38 public interface Dispatch<T> extends BindingProvider {
39
40     /** Invoke a service operation synchronously.
41     *
42     * The client is responsible for ensuring that the <code>msg</code> object
43     * when marshalled is formed according to the requirements of the protocol
44     * binding in use.
45     *
46     * @param msg An object that will form the message or payload of
47     * the message used to invoke the operation.

```

```

48     * @return The response message or message payload to the
49     *     operation invocation.
50     * @throws WebServiceException If a fault occurs during communication with
51     *     the service
52     * @throws WebServiceException If there is any error in the configuration of
53     *     the <code>Dispatch</code> instance
54     **/
55     public T invoke(T msg);
56
57     /** Invoke a service operation asynchronously. The
58     *     method returns without waiting for the response to the operation
59     *     invocation, the results of the operation are obtained by polling the
60     *     returned <code>Response</code>.
61     * <p>
62     *     The client is responsible for ensuring that the <code>msg</code> object
63     *     when marshalled is formed according to the requirements of the protocol
64     *     binding in use.
65     *
66     * @param msg An object that will form the message or payload of
67     *     the message used to invoke the operation.
68     * @return The response message or message payload to the
69     *     operation invocation.
70     * @throws WebServiceException If there is any error in the configuration of
71     *     the <code>Dispatch</code> instance
72     **/
73     public Response<T> invokeAsync(T msg);
74
75     /** Invoke a service operation asynchronously. The
76     *     method returns without waiting for the response to the operation
77     *     invocation, the results of the operation are communicated to the client
78     *     via the passed in <code>handler</code>.
79     * <p>
80     *     The client is responsible for ensuring that the <code>msg</code> object
81     *     when marshalled is formed according to the requirements of the protocol
82     *     binding in use.
83     *
84     * @param msg An object that will form the message or payload of
85     *     the message used to invoke the operation.
86     * @param handler The handler object that will receive the
87     *     response to the operation invocation.
88     * @return A <code>Future</code> object that may be used to check the status
89     *     of the operation invocation. This object MUST NOT be used to try to
90     *     obtain the results of the operation - the object returned from
91     *     <code>Future<?>.get()</code> is implementation dependent
92     *     and any use of it will result in non-portable behaviour.
93     * @throws WebServiceException If there is any error in the configuration of
94     *     the <code>Dispatch</code> instance
95     **/
96     public Future<?> invokeAsync(T msg, AsyncHandler<T> handler);
97
98     /** Invokes a service operation using the one-way
99     *     interaction mode. The operation invocation is logically non-blocking,
100    *     subject to the capabilities of the underlying protocol, no results

```

```
101     * are returned. When
102     * the protocol in use is SOAP/HTTP, this method MUST block until
103     * an HTTP response code has been received or an error occurs.
104     * <p>
105     * The client is responsible for ensuring that the <code>msg</code> object
106     * when marshalled is formed according to the requirements of the protocol
107     * binding in use.
108     *
109     * @param msg An object that will form the message or payload of
110     *     the message used to invoke the operation.
111     * @throws WebServiceException If there is any error in the configuration of
112     *     the <code>Dispatch</code> instance or if an error occurs during the
113     *     invocation.
114     **/
115     public void invokeOneWay(T msg);
116 }
```

## 27.7 Endpoint.java

Secuencia 266: javax/xml/ws/Endpoint.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.util.List;
29 import java.util.Map;
30 import javax.xml.ws.spi.Provider;
31 import javax.xml.ws.spi.http.HttpContext;
32 import javax.xml.ws.wsaddressing.W3CEndpointReference;
33 import org.w3c.dom.Element;
```

```

34
35
36 /**
37  * A Web service endpoint.
38  *
39  * <p>Endpoints are created using the static methods defined in this
40  * class. An endpoint is always tied to one <code>Binding</code>
41  * and one implementor, both set at endpoint creation time.
42  *
43  * <p>An endpoint is either in a published or an unpublished state.
44  * The <code>publish</code> methods can be used to start publishing
45  * an endpoint, at which point it starts accepting incoming requests.
46  * Conversely, the <code>stop</code> method can be used to stop
47  * accepting incoming requests and take the endpoint down.
48  * Once stopped, an endpoint cannot be published again.
49  *
50  * <p>An <code>Executor</code> may be set on the endpoint in order
51  * to gain better control over the threads used to dispatch incoming
52  * requests. For instance, thread pooling with certain parameters
53  * can be enabled by creating a <code>ThreadPoolExecutor</code> and
54  * registering it with the endpoint.
55  *
56  * <p>Handler chains can be set using the contained <code>Binding</code>.
57  *
58  * <p>An endpoint may have a list of metadata documents, such as WSDL
59  * and XMLSchema documents, bound to it. At publishing time, the
60  * JAX-WS implementation will try to reuse as much of that metadata
61  * as possible instead of generating new ones based on the annotations
62  * present on the implementor.
63  *
64  * @since JAX-WS 2.0
65  *
66  * @see javax.xml.ws.Binding
67  * @see javax.xml.ws.BindingType
68  * @see javax.xml.ws.soap.SOAPBinding
69  * @see java.util.concurrent.Executor
70  *
71  */
72 public abstract class Endpoint {
73
74     /** Standard property: name of WSDL service.
75      * <p>Type: javax.xml.namespace.QName
76      */
77     public static final String WSDL_SERVICE = "javax.xml.ws.wsd.service";
78
79     /** Standard property: name of WSDL port.
80      * <p>Type: javax.xml.namespace.QName
81      */
82     public static final String WSDL_PORT = "javax.xml.ws.wsd.port";
83
84
85     /**
86      * Creates an endpoint with the specified implementor object. If there is

```

```

87      * a binding specified via a BindingType annotation then it MUST be used else
88      * a default of SOAP 1.1 / HTTP binding MUST be used.
89      * <p>
90      * The newly created endpoint may be published by calling
91      * one of the {@link javax.xml.ws.Endpoint#publish(String)} and
92      * {@link javax.xml.ws.Endpoint#publish(Object)} methods.
93      *
94      *
95      * @param implementor The endpoint implementor.
96      *
97      * @return The newly created endpoint.
98      *
99      **/
100     public static Endpoint create(Object implementor) {
101         return create(null, implementor);
102     }
103
104     /**
105      * Creates an endpoint with the specified implementor object and web
106      * service features. If there is a binding specified via a BindingType
107      * annotation then it MUST be used else a default of SOAP 1.1 / HTTP
108      * binding MUST be used.
109      * <p>
110      * The newly created endpoint may be published by calling
111      * one of the {@link javax.xml.ws.Endpoint#publish(String)} and
112      * {@link javax.xml.ws.Endpoint#publish(Object)} methods.
113      *
114      *
115      * @param implementor The endpoint implementor.
116      * @param features A list of WebServiceFeature to configure on the
117      *     endpoint. Supported features not in the <code>features
118      *     </code> parameter will have their default values.
119      *
120      *
121      * @return The newly created endpoint.
122      * @since JAX-WS 2.2
123      *
124      */
125     public static Endpoint create(Object implementor, WebServiceFeature ... features) {
126         return create(null, implementor, features);
127     }
128
129     /**
130      * Creates an endpoint with the specified binding type and
131      * implementor object.
132      * <p>
133      * The newly created endpoint may be published by calling
134      * one of the {@link javax.xml.ws.Endpoint#publish(String)} and
135      * {@link javax.xml.ws.Endpoint#publish(Object)} methods.
136      *
137      *
138      * @param bindingId A URI specifying the binding to use. If the bindingID is
139      *     <code>null</code> and no binding is specified via a BindingType
140      *     annotation then a default SOAP 1.1 / HTTP binding MUST be used.

```

```

140     *
141     * @param implementor The endpoint implementor.
142     *
143     * @return The newly created endpoint.
144     *
145     **/
146     public static Endpoint create(String bindingId, Object implementor) {
147         return Provider.provider().createEndpoint(bindingId, implementor);
148     }
149
150     /**
151     * Creates an endpoint with the specified binding type,
152     * implementor object, and web service features.
153     * <p>
154     * The newly created endpoint may be published by calling
155     * one of the {@link javax.xml.ws.Endpoint#publish(String)} and
156     * {@link javax.xml.ws.Endpoint#publish(Object)} methods.
157     *
158     * @param bindingId A URI specifying the binding to use. If the bindingID is
159     * <code>null</code> and no binding is specified via a BindingType
160     * annotation then a default SOAP 1.1 / HTTP binding MUST be used.
161     *
162     * @param implementor The endpoint implementor.
163     *
164     * @param features A list of WebServiceFeature to configure on the
165     *     endpoint. Supported features not in the <code>features
166     *     </code> parameter will have their default values.
167     *
168     * @return The newly created endpoint.
169     * @since JAX-WS 2.2
170     */
171     public static Endpoint create(String bindingId, Object implementor, WebServiceFeature ...
172         features) {
173         return Provider.provider().createEndpoint(bindingId, implementor, features);
174     }
175
176     /**
177     * Returns the binding for this endpoint.
178     *
179     * @return The binding for this endpoint
180     **/
181     public abstract Binding getBinding();
182
183     /**
184     * Returns the implementation object for this endpoint.
185     *
186     * @return The implementor for this endpoint
187     **/
188     public abstract Object getImplementor();
189
190     /**
191     * Publishes this endpoint at the given address.
192     * The necessary server infrastructure will be created and

```

```

192     * configured by the JAX-WS implementation using some default configuration.
193     * In order to get more control over the server configuration, please
194     * use the {@link javax.xml.ws.Endpoint#publish(Object)} method instead.
195     *
196     * @param address A URI specifying the address to use. The address
197     *       MUST be compatible with the binding specified at the
198     *       time the endpoint was created.
199     *
200     * @throws java.lang.IllegalArgumentException
201     *       If the provided address URI is not usable
202     *       in conjunction with the endpoint's binding.
203     *
204     * @throws java.lang.IllegalStateException
205     *       If the endpoint has been published already or it has been stopped.
206     *
207     * @throws java.lang.SecurityException
208     *       If a <code>java.lang.SecurityManger</code>
209     *       is being used and the application doesn't have the
210     *       <code>WebServicePermission("publishEndpoint")</code> permission.
211     **/
212     public abstract void publish(String address);
213
214     /**
215     * Creates and publishes an endpoint for the specified implementor
216     * object at the given address.
217     * <p>
218     * The necessary server infrastructure will be created and
219     * configured by the JAX-WS implementation using some default configuration.
220     *
221     * In order to get more control over the server configuration, please
222     * use the {@link javax.xml.ws.Endpoint#create(String,Object)} and
223     * {@link javax.xml.ws.Endpoint#publish(Object)} methods instead.
224     *
225     * @param address A URI specifying the address and transport/protocol
226     *       to use. A http: URI MUST result in the SOAP 1.1/HTTP
227     *       binding being used. Implementations may support other
228     *       URI schemes.
229     * @param implementor The endpoint implementor.
230     *
231     * @return The newly created endpoint.
232     *
233     * @throws java.lang.SecurityException
234     *       If a <code>java.lang.SecurityManger</code>
235     *       is being used and the application doesn't have the
236     *       <code>WebServicePermission("publishEndpoint")</code> permission.
237     *
238     **/
239     public static Endpoint publish(String address, Object implementor) {
240         return Provider.provider().createAndPublishEndpoint(address, implementor);
241     }
242
243     /**
244     * Creates and publishes an endpoint for the specified implementor

```

```

245 * object at the given address. The created endpoint is configured
246 * with the web service features.
247 * <p>
248 * The necessary server infrastructure will be created and
249 * configured by the JAX-WS implementation using some default configuration.
250 *
251 * In order to get more control over the server configuration, please
252 * use the {@link javax.xml.ws.Endpoint#create(String,Object)} and
253 * {@link javax.xml.ws.Endpoint#publish(Object)} methods instead.
254 *
255 * @param address A URI specifying the address and transport/protocol
256 *       to use. A http: URI MUST result in the SOAP 1.1/HTTP
257 *       binding being used. Implementations may support other
258 *       URI schemes.
259 * @param implementor The endpoint implementor.
260 * @param features A list of WebServiceFeature to configure on the
261 *       endpoint. Supported features not in the <code>features
262 *       </code> parameter will have their default values.
263 * @return The newly created endpoint.
264 *
265 * @throws java.lang.SecurityException
266 *       If a <code>java.lang.SecurityManger</code>
267 *       is being used and the application doesn't have the
268 *       <code>WebServicePermission("publishEndpoint")</code> permission.
269 * @since JAX-WS 2.2
270 */
271 public static Endpoint publish(String address, Object implementor, WebServiceFeature ...
272     features) {
273     return Provider.provider().createAndPublishEndpoint(address, implementor, features);
274 }
275
276 /**
277 * Publishes this endpoint at the provided server context.
278 * A server context encapsulates the server infrastructure
279 * and addressing information for a particular transport.
280 * For a call to this method to succeed, the server context
281 * passed as an argument to it MUST be compatible with the
282 * endpoint's binding.
283 *
284 * @param serverContext An object representing a server
285 *       context to be used for publishing the endpoint.
286 *
287 * @throws java.lang.IllegalArgumentException
288 *       If the provided server context is not
289 *       supported by the implementation or turns
290 *       out to be unusable in conjunction with the
291 *       endpoint's binding.
292 *
293 * @throws java.lang.IllegalStateException
294 *       If the endpoint has been published already or it has been stopped.
295 *
296 * @throws java.lang.SecurityException

```



```

297     *      If a <code>java.lang.SecurityManger</code>
298     *      is being used and the application doesn't have the
299     *      <code>WebServicePermission("publishEndpoint")</code> permission.
300     **/
301     public abstract void publish(Object serverContext);
302
303     /**
304     * Publishes this endpoint at the provided server context.
305     * A server context encapsulates the server infrastructure
306     * and addressing information for a particular transport.
307     * For a call to this method to succeed, the server context
308     * passed as an argument to it MUST be compatible with the
309     * endpoint's binding.
310     *
311     * <p>
312     * This is meant for container developers to publish the
313     * the endpoints portably and not intended for the end
314     * developers.
315     *
316     *
317     * @param serverContext An object representing a server
318     *      context to be used for publishing the endpoint.
319     *
320     * @throws java.lang.IllegalArgumentException
321     *      If the provided server context is not
322     *      supported by the implementation or turns
323     *      out to be unusable in conjunction with the
324     *      endpoint's binding.
325     *
326     * @throws java.lang.IllegalStateException
327     *      If the endpoint has been published already or it has been stopped.
328     *
329     * @throws java.lang.SecurityException
330     *      If a <code>java.lang.SecurityManger</code>
331     *      is being used and the application doesn't have the
332     *      <code>WebServicePermission("publishEndpoint")</code> permission.
333     * @since JAX-WS 2.2
334     */
335     public void publish(HttpContext serverContext) {
336         throw new UnsupportedOperationException("JAX-WS 2.2 implementation must override this
337         default behaviour.");
338     }
339
340     /**
341     * Stops publishing this endpoint.
342     *
343     * If the endpoint is not in a published state, this method
344     * has no effect.
345     *
346     **/
347     public abstract void stop();
348
349     /**

```

```
349     * Returns true if the endpoint is in the published state.
350     *
351     * @return <code>true</code> if the endpoint is in the published state.
352     **/
353     public abstract boolean isPublished();
354
355     /**
356     * Returns a list of metadata documents for the service.
357     *
358     * @return <code>List<javax.xml.transform.Source></code> A list of metadata documents for
359     *         the service
360     **/
361     public abstract List<javax.xml.transform.Source> getMetadata();
362
363     /**
364     * Sets the metadata for this endpoint.
365     *
366     * @param metadata A list of XML document sources containing
367     *                 metadata information for the endpoint (e.g.
368     *                 WSDL or XML Schema documents)
369     *
370     * @throws java.lang.IllegalStateException If the endpoint
371     *         has already been published.
372     **/
373     public abstract void setMetadata(List<javax.xml.transform.Source> metadata);
374
375     /**
376     * Returns the executor for this <code>Endpoint</code> instance.
377     *
378     * The executor is used to dispatch an incoming request to
379     * the implementor object.
380     *
381     * @return The <code>java.util.concurrent.Executor</code> to be
382     *         used to dispatch a request.
383     *
384     * @see java.util.concurrent.Executor
385     **/
386     public abstract java.util.concurrent.Executor getExecutor();
387
388     /**
389     * Sets the executor for this <code>Endpoint</code> instance.
390     *
391     * The executor is used to dispatch an incoming request to
392     * the implementor object.
393     *
394     * If this <code>Endpoint</code> is published using the
395     * <code>publish(Object)</code> method and the specified server
396     * context defines its own threading behavior, the executor
397     * may be ignored.
398     *
399     * @param executor The <code>java.util.concurrent.Executor</code>
400     *                 to be used to dispatch a request.
401     *
```

```

401     * @throws SecurityException If the instance does not support
402     *       setting an executor for security reasons (e.g. the
403     *       necessary permissions are missing).
404     *
405     * @see java.util.concurrent.Executor
406     */
407     public abstract void setExecutor(java.util.concurrent.Executor executor);
408
409
410     /**
411     * Returns the property bag for this <code>Endpoint</code> instance.
412     *
413     * @return Map<String, Object> The property bag
414     *       associated with this instance.
415     */
416     public abstract Map<String, Object> getProperties();
417
418     /**
419     * Sets the property bag for this <code>Endpoint</code> instance.
420     *
421     * @param properties The property bag associated with
422     *       this instance.
423     */
424     public abstract void setProperties(Map<String, Object> properties);
425
426     /**
427     * Returns the <code>EndpointReference</code> associated with
428     * this <code>Endpoint</code> instance.
429     * <p>
430     * If the Binding for this <code>bindingProvider</code> is
431     * either SOAP1.1/HTTP or SOAP1.2/HTTP, then a
432     * <code>W3CEndpointReference</code> MUST be returned.
433     *
434     * @param referenceParameters Reference parameters to be associated with the
435     *       returned <code>EndpointReference</code> instance.
436     * @return EndpointReference of this <code>Endpoint</code> instance.
437     * If the returned <code>EndpointReference</code> is of type
438     * <code>W3CEndpointReference</code> then it MUST contain the
439     * the specified <code>referenceParameters</code>.
440
441     * @throws WebServiceException If any error in the creation of
442     *       the <code>EndpointReference</code> or if the <code>Endpoint</code> is
443     *       not in the published state.
444     * @throws UnsupportedOperationException If this <code>BindingProvider</code>
445     *       uses the XML/HTTP binding.
446     *
447     * @see W3CEndpointReference
448     *
449     * @since JAX-WS 2.1
450     */
451     public abstract EndpointReference getEndpointReference(Element... referenceParameters);
452
453

```

```

454 /**
455  * Returns the EndpointReference associated with
456  * this Endpoint instance.
457  *
458  * @param clazz Specifies the type of EndpointReference that MUST be returned.
459  * @param referenceParameters Reference parameters to be associated with the
460  * returned EndpointReference instance.
461  * @return EndpointReference of type clazz of this
462  * Endpoint instance.
463  * If the returned EndpointReference is of type
464  * W3CEndpointReference then it MUST contain the
465  * the specified referenceParameters.
466  *
467  * @throws WebServiceException If any error in the creation of
468  * the EndpointReference or if the Endpoint is
469  * not in the published state or if the clazz is not a supported
470  * EndpointReference type.
471  * @throws UnsupportedOperationException If this BindingProvider
472  * uses the XML/HTTP binding.
473  *
474  *
475  * @since JAX-WS 2.1
476  */
477 public abstract <T extends EndpointReference> T getEndpointReference(Class<T> clazz,
478     Element... referenceParameters);
479
480 /**
481  * By setting a EndpointContext, JAX-WS runtime knows about
482  * addresses of other endpoints in an application. If multiple endpoints
483  * share different ports of a WSDL, then the multiple port addresses
484  * are patched when the WSDL is accessed.
485  *
486  * <p>
487  * This needs to be set before publishing the endpoints.
488  *
489  * @param ctxt that is shared for multiple endpoints
490  * @throws java.lang.IllegalStateException
491  *     If the endpoint has been published already or it has been stopped.
492  *
493  * @since JAX-WS 2.2
494  */
495 public void setEndpointContext(EndpointContext ctxt) {
496     throw new UnsupportedOperationException("JAX-WS 2.2 implementation must override this
497         default behaviour.");
498 }

```

## 27.8 EndpointContext.java

Secuencia 267: javax/xml/ws/EndpointContext.java

```

1  /**
2  * Copyright (c) 2009, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import javax.xml.ws.Endpoint;
29 import java.util.Set;
30
31 /**
32  * <code>EndpointContext</code> allows multiple endpoints in an application
33  * to share any information. For example, servlet application's war may
34  * contain multiple endpoints and these endpoints can get addresses of each
35  * other by sharing this context. If multiple endpoints share different
36  * ports of a WSDL, then the multiple port addresses can be patched
37  * when the WSDL is accessed. It also allows all endpoints to share any
38  * other runtime information.
39  *
40  * <p>
41  * This needs to be set by using {@link Endpoint#setEndpointContext}
42  * before {@link Endpoint#publish} methods.
43  *
44  * @author Jitendra Kotamraju
45  * @since JAX-WS 2.2
46  */
47 public abstract class EndpointContext {
48
49     /**
50      * This gives list of endpoints in an application. For e.g in
51      * servlet container, a war file may contain multiple endpoints.
52      * In case of http, these endpoints are hosted on the same http
53      * server.
54      *
55      * @return list of all endpoints in an application
56      */
57 }
```

```

57     public abstract Set<Endpoint> getEndpoints();
58
59 }

```

## 27.9 EndpointReference.java

Secuencia 268: javax/xml/ws/EndpointReference.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import javax.xml.bind.annotation.XmlTransient;
29 import javax.xml.transform.Result;
30 import javax.xml.transform.Source;
31 import javax.xml.transform.stream.StreamResult;
32 import javax.xml.ws.spi.Provider;
33 import javax.xml.ws.wsaddressing.W3CEndpointReference;
34 import java.io.StringWriter;
35
36 /**
37 * This class represents an WS-Addressing EndpointReference
38 * which is a remote reference to a web service endpoint.
39 * See <a href="http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/">
40 * Web Services Addressing 1.0 - Core</a>
41 * for more information on WS-Addressing EndpointReferences.
42 * <p>
43 * This class is immutable as the typical web service developer
44 * need not be concerned with its contents. The web service
45 * developer should use this class strictly as a mechanism to
46 * reference a remote web service endpoint. See the {@link Service} APIs

```

```

47 * that clients can use to that utilize an <code>EndpointReference</code>.
48 * See the {@link javax.xml.ws.Endpoint}, and
49 * {@link javax.xml.ws.BindingProvider} APIs on how
50 * <code>EndpointReferences</code> can be created for published
51 * endpoints.
52 * <p>
53 * Concrete implementations of this class will represent
54 * an <code>EndpointReference</code> for a particular version of Addressing.
55 * For example the {@link W3CEndpointReference} is for use
56 * with W3C Web Services Addressing 1.0 - Core Recommendation.
57 * If JAX-WS implementors need to support different versions
58 * of addressing, they should write their own
59 * <code>EndpointReference</code> subclass for that version.
60 * This will allow a JAX-WS implementation to create
61 * a vendor specific <code>EndpointReferences</code> that the
62 * vendor can use to flag a different version of
63 * addressing.
64 * <p>
65 * Web service developers that wish to pass or return
66 * <code>EndpointReference</code> in Java methods in an
67 * SEI should use
68 * concrete instances of an <code>EndpointReference</code> such
69 * as the <code>W3CEndpointReference</code>. This way the
70 * schema mapped from the SEI will be more descriptive of the
71 * type of endpoint reference being passed.
72 * <p>
73 * JAX-WS implementors are expected to extract the XML info set
74 * from an <CODE>EndpointReference</CODE> using the
75 * <code>{@link EndpointReference#writeTo}</code>
76 * method.
77 * <p>
78 * JAXB will bind this class to xs:anyType. If a better binding
79 * is desired, web services developers should use a concrete
80 * subclass such as {@link W3CEndpointReference}.
81 *
82 * @see W3CEndpointReference
83 * @see Service
84 * @since JAX-WS 2.1
85 */
86 @XmlTransient // to treat this class like Object as far as databinding is concerned (proposed JAXB
    2.1 feature)
87 public abstract class EndpointReference {
88     //
89     //Default constructor to be only called by derived types.
90     //
91     protected EndpointReference(){}
92
93     /**
94      * Factory method to read an EndpointReference from the info set contained in
95      * <code>eprInfoSet</code>. This method delegates to the vendor specific
96      * implementation of the {@link javax.xml.ws.spi.Provider#readEndpointReference} method.
97      *
98      * @param eprInfoSet The <code>EndpointReference</code> info set to be unmarshalled

```

```

99      *
100     * @return the EndpointReference unmarshalled from <code>eprInfoSet</code>
101     *     never <code>null</code>
102     * @throws WebServiceException
103     *     if an error occurs while creating the
104     *     <code>EndpointReference</code> from the <CODE>eprInfoSet</CODE>
105     * @throws java.lang.IllegalArgumentException
106     *     if the <code>null</code> <code>eprInfoSet</code> value is given.
107     */
108     public static EndpointReference readFrom(Source eprInfoSet) {
109         return Provider.provider().readEndpointReference(eprInfoSet);
110     }
111
112     /**
113     * write this <code>EndpointReference</code> to the specified infoSet format
114     *
115     * @param result for writing infoSet
116     * @throws WebServiceException
117     *     if there is an error writing the
118     *     <code>EndpointReference</code> to the specified <code>result</code>.
119     *
120     * @throws java.lang.IllegalArgumentException
121     *     If the <code>null</code> <code>result</code> value is given.
122     */
123     public abstract void writeTo(Result result);
124
125
126     /**
127     * The <code>getPort</code> method returns a proxy. If there
128     * are any reference parameters in the
129     * <code>EndpointReference</code> instance, then those reference
130     * parameters MUST appear as SOAP headers, indicating them to be
131     * reference parameters, on all messages sent to the endpoint.
132     * The parameter <code>serviceEndpointInterface</code> specifies
133     * the service endpoint interface that is supported by the
134     * returned proxy.
135     * The <code>EndpointReference</code> instance specifies the
136     * endpoint that will be invoked by the returned proxy.
137     * In the implementation of this method, the JAX-WS
138     * runtime system takes the responsibility of selecting a protocol
139     * binding (and a port) and configuring the proxy accordingly from
140     * the WSDL Metadata from this <code>EndpointReference</code> or from
141     * annotations on the <code>serviceEndpointInterface</code>. For this method
142     * to successfully return a proxy, WSDL metadata MUST be available and the
143     * <code>EndpointReference</code> instance MUST contain an implementation understood
144     * <code>serviceName</code> metadata.
145     * <p>
146     * Because this port is not created from a <code>Service</code> object, handlers
147     * will not automatically be configured, and the <code>HandlerResolver</code>
148     * and <code>Executor</code> cannot be get or set for this port. The
149     * <code>BindingProvider().getBinding().setHandlerChain()</code>
150     * method can be used to manually configure handlers for this port.
151     *

```



```

152  *
153  * @param serviceEndpointInterface Service endpoint interface
154  * @param features An array of <code>WebServiceFeatures</code> to configure on the
155  *                proxy. Supported features not in the <code>features
156  *                </code> parameter will have their default values.
157  * @return Object Proxy instance that supports the
158  *                specified service endpoint interface
159  * @throws WebServiceException
160  *         <UL>
161  *         <LI>If there is an error during creation
162  *         of the proxy
163  *         <LI>If there is any missing WSDL metadata
164  *         as required by this method
165  *         <LI>If this
166  *         <code>endpointReference</code>
167  *         is invalid
168  *         <LI>If an illegal
169  *         <code>serviceEndpointInterface</code>
170  *         is specified
171  *         <LI>If a feature is enabled that is not compatible with
172  *         this port or is unsupported.
173  *         </UL>
174  *
175  * @see java.lang.reflect.Proxy
176  * @see WebServiceFeature
177  **/
178  public <T> T getPort(Class<T> serviceEndpointInterface,
179                      WebServiceFeature... features) {
180      return Provider.provider().getPort(this, serviceEndpointInterface,
181                                         features);
182  }
183
184  /**
185   * Displays EPR info set for debugging convenience.
186   */
187  public String toString() {
188      StringWriter w = new StringWriter();
189      writeTo(new StreamResult(w));
190      return w.toString();
191  }
192  }

```

## 27.10 FaultAction.java

Secuencia 269: javax/xml/ws/FaultAction.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.ElementType;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.RetentionPolicy;
32 import java.lang.annotation.Target;
33
34 /**
35  * The FaultAction annotation is used inside an @link Action
36  * annotation to allow an explicit association of a WS-Addressing
37  * Action message addressing property with the fault
38  * messages of the WSDL operation mapped from the exception class.
39  * 

40  * The wsam:Action attribute value in the fault
41  * message in the generated WSDL operation mapped for className
42  * class is equal to the corresponding value in the FaultAction.
43  * For the exact computation of wsam:Action values for the
44  * fault messages, refer to the algorithm in the JAX-WS specification.
45  *
46  * 

47  * Example 1: Specify explicit values for Action message addressing
48  * property for the input, output and fault message
49  * if the Java method throws only one service specific exception.
50  *
51  * 

```

52  * &#64;WebService(targetNamespace="http://example.com/numbers")
53  * public class AddNumbersImpl {
54  *     &#64;Action(
55  *         fault = {
56  *             <b>&#64;FaultAction(className=AddNumbersException.class, value="http://example.com/
57  *                 faultAction")</b>
58  *         })
59  *     public int addNumbers(int number1, int number2)
60  *         throws AddNumbersException {
61  *         return number1 + number2;
62  *     }
63  * }
```


```

```

61 *     }
62 * }
63 * </pre>
64 *
65 * The generated WSDL looks like:
66 *
67 * <pre>
68 *     <definitions targetNamespace="http://example.com/numbers" ...>
69 *         ...
70 *         <portType name="AddNumbersPortType">
71 *             <operation name="AddNumbers">
72 *                 ...
73 *                 <fault message="tns:AddNumbersException" name="AddNumbersException"
74 *                     <b>wsam:Action="http://example.com/faultAction"</b>/>
75 *             </operation>
76 *         </portType>
77 *         ...
78 *     </definitions>
79 * </pre>
80 *
81 * <p>
82 * Example 2: Here is an example that shows if the explicit value for <code>Action</code>
83 * message addressing property for the service specific exception is not present.
84 *
85 * <pre>
86 * &#64;WebService(targetNamespace="http://example.com/numbers")
87 * public class AddNumbersImpl {
88 *     public int addNumbers(int number1, int number2)
89 *         throws AddNumbersException {
90 *         return number1 + number2;
91 *     }
92 * }
93 * </pre>
94 *
95 * The generated WSDL looks like:
96 *
97 * <pre>
98 *     <definitions targetNamespace="http://example.com/numbers" ...>
99 *         ...
100 *         <portType name="AddNumbersPortType">
101 *             <operation name="AddNumbers">
102 *                 ...
103 *                 <fault message="tns:addNumbersFault" name="InvalidNumbers"
104 *                     <b>wsam:Action="http://example.com/numbers/AddNumbersPortType/AddNumbers/Fault/
105 *                         AddNumbersException"</b>/>
106 *             </operation>
107 *         </portType>
108 *         ...
109 *     </definitions>
110 * </pre>
111 *
112 * <p>
113 * Example 3: Here is an example that shows how to specify explicit values for <code>Action</code>

```

```

113 * message addressing property if the Java method throws more than one service specific exception.
114 *
115 * <pre>
116 * &#64;WebService(targetNamespace="http://example.com/numbers")
117 * public class AddNumbersImpl {
118 *     &#64;Action(
119 *         fault = {
120 *             <b>&#64;FaultAction(className=AddNumbersException.class, value="http://example.com/
addFaultAction"),
121 *             &#64;FaultAction(className=TooBigNumbersException.class, value="http://example.com/
toobigFaultAction")</b>
122 *         })
123 *     public int addNumbers(int number1, int number2)
124 *         throws AddNumbersException, TooBigNumbersException {
125 *         return number1 + number2;
126 *     }
127 * }
128 * </pre>
129 *
130 * The generated WSDL looks like:
131 *
132 * <pre>
133 * &lt;definitions targetNamespace="http://example.com/numbers" ...>
134 *     ...
135 *     &lt;portType name="AddNumbersPortType">
136 *         &lt;operation name="AddNumbers">
137 *             ...
138 *             &lt;fault message="tns:addNumbersFault" name="AddNumbersException"
139 *                 <b>wsam:Action="http://example.com/addFaultAction"</b>>
140 *             &lt;fault message="tns:tooBigNumbersFault" name="TooBigNumbersException"
141 *                 <b>wsam:Action="http://example.com/toobigFaultAction"</b>>
142 *             &lt;/operation>
143 *         &lt;/portType>
144 *     ...
145 * &lt;/definitions>
146 * </pre>
147 *
148 * @since JAX-WS 2.1
149 */
150
151 @Documented
152 @Retention(RetentionPolicy.RUNTIME)
153 @Target(ElementType.METHOD)
154 public @interface FaultAction {
155     /**
156      * Name of the exception class
157      */
158     Class<? extends Exception> className();
159
160     /**
161      * Value of WS-Addressing <code>Action</code> message addressing property for the exception
162      */
163     String value() default "";

```

164 }

**27.11 Holder.java**

Secuencia 270: javax.xml.ws/Holder.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.io.Serializable;
29
30 /**
31  * Holds a value of type T.
32  *
33  * @since JAX-WS 2.0
34  */
35 public final class Holder<T> implements Serializable {
36
37     private static final long serialVersionUID = 2623699057546497185L;
38
39     /**
40      * The value contained in the holder.
41      */
42     public T value;
43
44     /**
45      * Creates a new holder with a null value.
46      */
47     public Holder() {
48 }
```

```

49
50  /**
51   * Create a new holder with the specified value.
52   *
53   * @param value The value to be stored in the holder.
54   */
55  public Holder(T value) {
56      this.value = value;
57  }
58 }

```

## 27.12 LogicalMessage.java

Secuencia 271: javax/xml/ws/LogicalMessage.java

```

1  /*
2   * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws;
27
28 import javax.xml.transform.Source;
29 import javax.xml.bind.JAXBContext;
30
31 /** The <code>LogicalMessage</code> interface represents a
32  * protocol agnostic XML message and contains methods that
33  * provide access to the payload of the message.
34  *
35  * @since JAX-WS 2.0
36  */
37 public interface LogicalMessage {
38
39     /** Gets the message payload as an XML source, may be called

```

```
40 * multiple times on the same LogicalMessage instance, always
41 * returns a new <code>Source</code> that may be used to retrieve the entire
42 * message payload.
43 *
44 * <p>If the returned <code>Source</code> is an instance of
45 * <code>DOMSource</code>, then
46 * modifications to the encapsulated DOM tree change the message
47 * payload in-place, there is no need to subsequently call
48 * <code>setPayload</code>. Other types of <code>Source</code> provide only
49 * read access to the message payload.
50 *
51 * @return The contained message payload; returns <code>null</code> if no
52 *         payload is present in this message.
53 **/
54 public Source getPayload();
55
56 /** Sets the message payload
57 *
58 * @param payload message payload
59 * @throws WebServiceException If any error during the setting
60 *         of the payload in this message
61 * @throws java.lang.UnsupportedOperationException If this
62 *         operation is not supported
63 **/
64 public void setPayload(Source payload);
65
66 /** Gets the message payload as a JAXB object. Note that there is no
67 * connection between the returned object and the message payload,
68 * changes to the payload require calling <code>setPayload</code>.
69 *
70 * @param context The JAXBContext that should be used to unmarshall
71 *         the message payload
72 * @return The contained message payload; returns <code>null</code> if no
73 *         payload is present in this message
74 * @throws WebServiceException If an error occurs when using a supplied
75 *         JAXBContext to unmarshall the payload. The cause of
76 *         the WebServiceException is the original JAXBException.
77 **/
78 public Object getPayload(JAXBContext context);
79
80 /** Sets the message payload
81 *
82 * @param payload message payload
83 * @param context The JAXBContext that should be used to marshall
84 *         the payload
85 * @throws java.lang.UnsupportedOperationException If this
86 *         operation is not supported
87 * @throws WebServiceException If an error occurs when using the supplied
88 *         JAXBContext to marshall the payload. The cause of
89 *         the WebServiceException is the original JAXBException.
90 **/
91 public void setPayload(Object payload, JAXBContext context);
92 }
```

**27.13** ProtocolException.java

Secuencia 272: javax/xml/ws/ProtocolException.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 /** The <code>ProtocolException</code> class is a
29 * base class for exceptions related to a specific protocol binding. Subclasses
30 * are used to communicate protocol level fault information to clients and may
31 * be used on the server to control the protocol specific fault representation.
32 *
33 * @since JAX-WS 2.0
34 */
35 public class ProtocolException extends WebServiceException {
36     /**
37      * Constructs a new protocol exception with <code>null</code> as its detail message. The
38      * cause is not initialized, and may subsequently be initialized by a call
39      * to <code>Throwable.initCause(java.lang.Throwable)</code>.
40      */
41     public ProtocolException() {
42         super();
43     }
44
45     /**
46      * Constructs a new protocol exception with the specified detail message.
47      * The cause is not initialized, and may subsequently be initialized by a
48      * call to <code>Throwable.initCause(java.lang.Throwable)</code>.
49      *
50      * @param message the detail message. The detail message is saved for later
```



```

51     * retrieval by the Throwable.getMessage() method.
52     */
53     public ProtocolException(String message) {
54         super(message);
55     }
56
57     /**
58     * Constructs a new runtime exception with the specified detail message and
59     * cause.
60     *
61     * Note that the detail message associated with cause is not automatically
62     * incorporated in this runtime exception's detail message.
63     *
64     * @param message the detail message (which is saved for later retrieval by
65     * the Throwable.getMessage() method).
66     * @param cause the cause (which is saved for later retrieval by the
67     * <code>Throwable.getCause()</code> method). (A <code>null</code> value is permitted, and
        indicates
68     * that the cause is nonexistent or unknown.)
69     */
70     public ProtocolException(String message, Throwable cause) {
71         super(message, cause);
72     }
73
74     /**
75     * Constructs a new runtime exception with the specified cause and a detail
76     * message of <code>(cause==null ? null : cause.toString())</code> (which typically
77     * contains the class and detail message of cause). This constructor is
78     * useful for runtime exceptions that are little more than wrappers for
79     * other throwables.
80     *
81     * @param cause the cause (which is saved for later retrieval by the
82     * <code>Throwable.getCause()</code> method). (A <code>null</code> value is permitted, and
        indicates
83     * that the cause is nonexistent or unknown.)
84     */
85     public ProtocolException(Throwable cause) {
86         super(cause);
87     }
88 }

```

## 27.14 Provider.java

Secuencia 273: javax/xml/ws/Provider.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```

```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws;
27
28 /**
29  * <p>Service endpoints may implement the <code>Provider</code>
30  * interface as a dynamic alternative to an SEI.
31  *
32  * <p>Implementations are required to support <code>Provider<Source></code>,
33  * <code>Provider<SOAPMessage></code> and
34  * <code>Provider<DataSource></code>, depending on the binding
35  * in use and the service mode.
36  *
37  * <p>The <code>ServiceMode</code> annotation can be used to control whether
38  * the <code>Provider</code> instance will receive entire protocol messages
39  * or just message payloads.
40  *
41  * @since JAX-WS 2.0
42  *
43  * @see javax.xml.transform.Source
44  * @see javax.xml.soap.SOAPMessage
45  * @see javax.xml.ws.ServiceMode
46  */
47 public interface Provider<T> {
48
49     /** Invokes an operation according to the contents of the request
50      * message.
51      *
52      * @param request The request message or message payload.
53      * @return The response message or message payload. May be <code>null</code> if
54      * there is no response.
55      * @throws WebServiceException If there is an error processing request.
56      * The cause of the <code>WebServiceException</code> may be set to a subclass
57      * of <code>ProtocolException</code> to control the protocol level
58      * representation of the exception.
59      * @see javax.xml.ws.handler.MessageContext
60      * @see javax.xml.ws.ProtocolException
61      */
62     public T invoke(T request);

```

63 }

**27.15 RequestWrapper.java**

Secuencia 274: javax/xml/ws/RequestWrapper.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.ElementType;
32 import java.lang.annotation.RetentionPolicy;
33
34 /**
35  * Used to annotate methods in the Service Endpoint Interface with the request
36  * wrapper bean to be used at runtime. The default value of the <code>localName</code> is
37  * the <code>operationName</code>, as defined in <code>WebMethod</code> annotation and the
38  * <code>targetNamespace</code> is the target namespace of the SEI.
39  * <p> When starting from Java this annotation is used resolve
40  * overloading conflicts in document literal mode. Only the <code>className</code>
41  * is required in this case.
42  *
43  * @since JAX-WS 2.0
44  */
45
46 @Target(ElementType.METHOD)
47 @Retention(RetentionPolicy.RUNTIME)
48 @Documented
```

```

49 public @interface RequestWrapper {
50     /**
51      * Element's local name.
52      */
53     public String localName() default "";
54
55     /**
56      * Element's namespace name.
57      */
58     public String targetNamespace() default "";
59
60     /**
61      * Request wrapper bean name.
62      */
63     public String className() default "";
64
65     /**
66      * wsdl:part name for the wrapper part
67      *
68      * @since JAX-WS 2.2
69      */
70     public String partName() default "";
71
72 }

```

## 27.16 RespectBinding.java

Secuencia 275: javax/xml/ws/RespectBinding.java

```

1  /*
2   * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25

```

```

26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.ElementType;
31 import java.lang.annotation.Retention;
32 import java.lang.annotation.RetentionPolicy;
33 import javax.xml.ws.spi.WebServiceFeatureAnnotation;
34
35
36 /**
37  * This feature clarifies the use of the <code>wsdl:binding</code>
38  * in a JAX-WS runtime.
39  * <p>
40  * This annotation MUST only be used in conjunction the
41  * <code>javax.jws.WebService</code>, {@link WebServiceProvider},
42  * {@link WebServiceRef} annotations.
43  * When used with the <code>javax.jws.WebService</code> annotation this
44  * annotation MUST only be used on the service endpoint implementation
45  * class.
46  * When used with a <code>WebServiceRef</code> annotation, this annotation
47  * MUST only be used when a proxy instance is created. The injected SEI
48  * proxy, and endpoint MUST honor the values of the <code>RespectBinding</code>
49  * annotation.
50  * <p>
51  *
52  * This annotation's behaviour is defined by the corresponding feature
53  * {@link RespectBindingFeature}.
54  *
55  * @see RespectBindingFeature
56  *
57  * @since JAX-WS 2.1
58  */
59 @Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
60 @Retention(RetentionPolicy.RUNTIME)
61 @Documented
62 @WebServiceFeatureAnnotation(id=RespectBindingFeature.ID, bean=RespectBindingFeature.class)
63 public @interface RespectBinding {
64     /**
65      * Specifies if this feature is enabled or disabled.
66      */
67     boolean enabled() default true;
68 }

```

## 27.17 RespectBindingFeature.java

Secuencia 276: javax/xml/ws/RespectBindingFeature.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import javax.xml.ws.soap.AddressingFeature;
29
30 /**
31  * This feature clarifies the use of the wsdl:binding
32  * in a JAX-WS runtime.
33  *
34  * This feature can be used during the creation of SEI proxy, and
35  * {@link Dispatch} instances on the client side and {@link Endpoint}
36  * instances on the server side. This feature cannot be used for {@link Service}
37  * instance creation on the client side.
38  * <p>
39  * This feature is only useful with web services that have an
40  * associated WSDL. Enabling this feature requires that a JAX-WS
41  * implementation inspect the wsdl:binding for an
42  * endpoint at runtime to make sure that all wsdl:extensions
43  * that have the required attribute set to true
44  * are understood and are being used.
45  * <p>
46  * The following describes the affects of this feature with respect
47  * to be enabled or disabled:
48  * <ul>
49  * <li> ENABLED: In this Mode, a JAX-WS runtime MUST assure that all
50  * required wsdl:binding extensions(including policies) are
51  * either understood and used by the runtime, or explicitly disabled by the
52  * web service application. A web service can disable a particular
53  * extension if there is a corresponding {@link WebServiceFeature} or annotation.
54  * Similarly, a web service client can disable
55  * particular extension using the corresponding WebServiceFeature while
56  * creating a proxy or Dispatch instance.
57  * The runtime MUST also make sure that binding of
58  * SEI parameters/return values respect the wsdl:binding.
59  * With this feature enabled, if a required (wsdl:required="true")
```

```

60 * <code>wsdl:binding</code> extension is in the WSDL and it is not
61 * supported by a JAX-WS runtime and it has not
62 * been explicitly turned off by the web service developer, then
63 * that JAX-WS runtime MUST behave appropriately based on whether it is
64 * on the client or server:
65 * <ul>
66 *   <li>Client: runtime MUST throw a
67 *   {@link WebServiceException} no sooner than when one of the methods
68 *   above is invoked but no later than the first invocation of an endpoint
69 *   operation.
70 *   <li>Server: throw a {@link WebServiceException} and the endpoint MUST fail to deploy
71 * </ul>
72 *
73 * <li> DISABLED: In this Mode, an implementation may choose whether
74 * to inspect the <code>wsdl:binding</code> or not and to what degree
75 * the <code>wsdl:binding</code> will be inspected. For example,
76 * one implementation may choose to behave as if this feature is enabled,
77 * another implementation may only choose to verify the SEI's
78 * parameter/return type bindings.
79 * </ul>
80 *
81 * @see AddressingFeature
82 *
83 * @since JAX-WS 2.1
84 */
85 public final class RespectBindingFeature extends WebServiceFeature {
86     /**
87     *
88     * Constant value identifying the RespectBindingFeature
89     */
90     public static final String ID = "javax.xml.ws.RespectBindingFeature";
91
92
93     /**
94     * Creates an <code>RespectBindingFeature</code>.
95     * The instance created will be enabled.
96     */
97     public RespectBindingFeature() {
98         this.enabled = true;
99     }
100
101     /**
102     * Creates an RespectBindingFeature
103     *
104     * @param enabled specifies whether this feature should
105     * be enabled or not.
106     */
107     public RespectBindingFeature(boolean enabled) {
108         this.enabled = enabled;
109     }
110
111     /**
112     * {@inheritDoc}

```

```

113     */
114     public String getID() {
115         return ID;
116     }
117 }

```

## 27.18 Response.java

Secuencia 277: javax/xml/ws/Response.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.util.Map;
29 import java.util.concurrent.Future;
30
31 /** The Response interface provides methods used to obtain the
32  * payload and context of a message sent in response to an operation
33  * invocation.
34  *
35  * <p>For asynchronous operation invocations it provides additional methods
36  * to check the status of the request. The get(...) methods may
37  * throw the standard
38  * set of exceptions and their cause may be a RemoteException or a
39  * {@link WebServiceException} that represents the error that occurred during the
40  * asynchronous method invocation.</p>
41  *
42  * @since JAX-WS 2.0
43  */
44 public interface Response<T> extends Future<T> {

```



```

45     /** Gets the contained response context.
46     *
47     * @return The contained response context. May be <code>null</code> if a
48     * response is not yet available.
49     *
50     */
51     Map<String,Object> getContext();
52 }

```

## 27.19 ResponseWrapper.java

Secuencia 278: javax/xml/ws/ResponseWrapper.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.ElementType;
32 import java.lang.annotation.RetentionPolicy;
33 /**
34  * Used to annotate methods in the Service Endpoint Interface with the response
35  * wrapper bean to be used at runtime. The default value of the <code>localName</code> is
36  * the <code>operationName</code> as defined in <code>WebMethod</code> annotation appended with
37  * <code>Response</code> and the <code>targetNamespace</code> is the target namespace of the SEI.
38  * <p> When starting from Java this annotation is used resolve
39  * overloading conflicts in document literal mode. Only the <code>className</code>
40  * is required in this case.
41  *

```

```

42  * @since JAX-WS 2.0
43  **/
44
45  @Target(ElementType.METHOD)
46  @Retention(RetentionPolicy.RUNTIME)
47  @Documented
48  public @interface ResponseWrapper {
49
50      /**
51       * Element's local name.
52       */
53      public String localName() default "";
54
55      /**
56       * Element's namespace name.
57       */
58      public String targetNamespace() default "";
59
60      /**
61       * Response wrapper bean name.
62       */
63      public String className() default "";
64
65      /**
66       * wsdl:part name for the wrapper part
67       *
68       * @since JAX-WS 2.2
69       */
70      public String partName() default "";
71
72  }

```

## 27.20 Service.java

Secuencia 279: javax/xml/ws/Service.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *

```

```

19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.ws;
27
28  import javax.xml.namespace.QName;
29  import java.util.Iterator;
30  import javax.xml.ws.handler.HandlerResolver;
31  import javax.xml.bind.JAXBContext;
32  import javax.xml.ws.spi.ServiceDelegate;
33  import javax.xml.ws.spi.Provider;
34
35  /**
36   * <code>Service</code> objects provide the client view of a Web service.
37   * <p><code>Service</code> acts as a factory of the following:
38   * <ul>
39   * <li>Proxies for a target service endpoint.</li>
40   * <li>Instances of {@link javax.xml.ws.Dispatch} for
41   *     dynamic message-oriented invocation of a remote
42   *     operation.
43   * </li>
44   * </ul>
45   *
46   * <p>The ports available on a service can be enumerated using the
47   * <code>getPorts</code> method. Alternatively, you can pass a
48   * service endpoint interface to the unary <code>getPort</code> method
49   * and let the runtime select a compatible port.
50   *
51   * <p>Handler chains for all the objects created by a <code>Service</code>
52   * can be set by means of a <code>HandlerResolver</code>.
53   *
54   * <p>An <code>Executor</code> may be set on the service in order
55   * to gain better control over the threads used to dispatch asynchronous
56   * callbacks. For instance, thread pooling with certain parameters
57   * can be enabled by creating a <code>ThreadPoolExecutor</code> and
58   * registering it with the service.
59   *
60   * @since JAX-WS 2.0
61   *
62   * @see javax.xml.ws.spi.Provider
63   * @see javax.xml.ws.handler.HandlerResolver
64   * @see java.util.concurrent.Executor
65   */
66  public class Service {
67
68      private ServiceDelegate delegate;
69
70      /**
71       * The orientation of a dynamic client or service. <code>MESSAGE</code> provides
72       * access to entire protocol message, <code>PAYLOAD</code> to protocol message

```

```

72     * payload only.
73     **/
74     public enum Mode { MESSAGE, PAYLOAD }
75
76     protected Service(java.net.URL wsdlDocumentLocation, QName serviceName) {
77         delegate = Provider.provider().createServiceDelegate(wsdlDocumentLocation,
78             serviceName,
79             this.getClass());
80     }
81
82     protected Service(java.net.URL wsdlDocumentLocation, QName serviceName, WebServiceFeature ...
83         features) {
84         delegate = Provider.provider().createServiceDelegate(wsdlDocumentLocation,
85             serviceName,
86             this.getClass(), features);
87     }
88
89     /**
90     * The <code>getPort</code> method returns a proxy. A service client
91     * uses this proxy to invoke operations on the target
92     * service endpoint. The <code>serviceEndpointInterface</code>
93     * specifies the service endpoint interface that is supported by
94     * the created dynamic proxy instance.
95     *
96     * @param portName Qualified name of the service endpoint in
97     *                 the WSDL service description.
98     * @param serviceEndpointInterface Service endpoint interface
99     *                 supported by the dynamic proxy instance.
100    * @return Object Proxy instance that
101    *         supports the specified service endpoint
102    *         interface.
103    * @throws WebServiceException This exception is thrown in the
104    *         following cases:
105    *         <UL>
106    *         <LI>If there is an error in creation of
107    *             the proxy.
108    *         <LI>If there is any missing WSDL metadata
109    *             as required by this method.
110    *         <LI>If an illegal
111    *             <code>serviceEndpointInterface</code>
112    *             or <code>portName</code> is specified.
113    *         </UL>
114    * @see java.lang.reflect.Proxy
115    * @see java.lang.reflect.InvocationHandler
116    **/
117     public <T> T getPort(QName portName,
118         Class<T> serviceEndpointInterface) {
119         return delegate.getPort(portName, serviceEndpointInterface);
120     }
121
122     /**
123     * The <code>getPort</code> method returns a proxy. A service client

```

```

124 * uses this proxy to invoke operations on the target
125 * service endpoint. The <code>serviceEndpointInterface</code>
126 * specifies the service endpoint interface that is supported by
127 * the created dynamic proxy instance.
128 *
129 * @param portName Qualified name of the service endpoint in
130 *                 the WSDL service description.
131 * @param serviceEndpointInterface Service endpoint interface
132 *                 supported by the dynamic proxy instance.
133 * @param features A list of WebServiceFeatures to configure on the
134 *                 proxy. Supported features not in the <code>features
135 *                 </code> parameter will have their default values.
136 * @return Object Proxy instance that
137 *         supports the specified service endpoint
138 *         interface.
139 * @throws WebServiceException This exception is thrown in the
140 *         following cases:
141 *         <UL>
142 *         <LI>If there is an error in creation of
143 *         the proxy.
144 *         <LI>If there is any missing WSDL metadata
145 *         as required by this method.
146 *         <LI>If an illegal
147 *         <code>serviceEndpointInterface</code>
148 *         or <code>portName</code> is specified.
149 *         <LI>If a feature is enabled that is not compatible
150 *         with this port or is unsupported.
151 *         </UL>
152 * @see java.lang.reflect.Proxy
153 * @see java.lang.reflect.InvocationHandler
154 * @see WebServiceFeature
155 *
156 * @since JAX-WS 2.1
157 */
158 public <T> T getPort(QName portName,
159                     Class<T> serviceEndpointInterface, WebServiceFeature... features) {
160     return delegate.getPort(portName, serviceEndpointInterface, features);
161 }
162
163
164 /**
165 * The <code>getPort</code> method returns a proxy. The parameter
166 * <code>serviceEndpointInterface</code> specifies the service
167 * endpoint interface that is supported by the returned proxy.
168 * In the implementation of this method, the JAX-WS
169 * runtime system takes the responsibility of selecting a protocol
170 * binding (and a port) and configuring the proxy accordingly.
171 * The returned proxy should not be reconfigured by the client.
172 *
173 * @param serviceEndpointInterface Service endpoint interface.
174 * @return Object instance that supports the
175 *         specified service endpoint interface.
176 * @throws WebServiceException

```

```

177      *          <UL>
178      *          <LI>If there is an error during creation
179      *          of the proxy.
180      *          <LI>If there is any missing WSDL metadata
181      *          as required by this method.
182      *          <LI>If an illegal
183      *          <code>serviceEndpointInterface</code>
184      *          is specified.
185      *          </UL>
186      **/
187      public <T> T getPort(Class<T> serviceEndpointInterface) {
188          return delegate.getPort(serviceEndpointInterface);
189      }
190
191
192      /**
193      * The <code>getPort</code> method returns a proxy. The parameter
194      * <code>serviceEndpointInterface</code> specifies the service
195      * endpoint interface that is supported by the returned proxy.
196      * In the implementation of this method, the JAX-WS
197      * runtime system takes the responsibility of selecting a protocol
198      * binding (and a port) and configuring the proxy accordingly.
199      * The returned proxy should not be reconfigured by the client.
200      *
201      * @param serviceEndpointInterface Service endpoint interface.
202      * @param features A list of WebServiceFeatures to configure on the
203      * proxy. Supported features not in the <code>features
204      * </code> parameter will have their default values.
205      * @return Object instance that supports the
206      * specified service endpoint interface.
207      * @throws WebServiceException
208      *          <UL>
209      *          <LI>If there is an error during creation
210      *          of the proxy.
211      *          <LI>If there is any missing WSDL metadata
212      *          as required by this method.
213      *          <LI>If an illegal
214      *          <code>serviceEndpointInterface</code>
215      *          is specified.
216      *          <LI>If a feature is enabled that is not compatible
217      *          with this port or is unsupported.
218      *          </UL>
219      *
220      * @see WebServiceFeature
221      *
222      * @since JAX-WS 2.1
223      **/
224      public <T> T getPort(Class<T> serviceEndpointInterface,
225          WebServiceFeature... features) {
226          return delegate.getPort(serviceEndpointInterface, features);
227      }
228
229

```

```

230  /**
231   * The getPort method returns a proxy.
232   * The parameter endpointReference specifies the
233   * endpoint that will be invoked by the returned proxy. If there
234   * are any reference parameters in the
235   * endpointReference, then those reference
236   * parameters MUST appear as SOAP headers, indicating them to be
237   * reference parameters, on all messages sent to the endpoint.
238   * The endpointReference's address MUST be used
239   * for invocations on the endpoint.
240   * The parameter serviceEndpointInterface specifies
241   * the service endpoint interface that is supported by the
242   * returned proxy.
243   * In the implementation of this method, the JAX-WS
244   * runtime system takes the responsibility of selecting a protocol
245   * binding (and a port) and configuring the proxy accordingly from
246   * the WSDL associated with this Service instance or
247   * from the metadata from the endpointReference.
248   * If this Service instance has a WSDL and
249   * the endpointReference metadata
250   * also has a WSDL, then the WSDL from this instance MUST be used.
251   * If this Service instance does not have a WSDL and
252   * the endpointReference does have a WSDL, then the
253   * WSDL from the endpointReference MAY be used.
254   * The returned proxy should not be reconfigured by the client.
255   * If this Service instance has a known proxy
256   * port that matches the information contained in
257   * the WSDL,
258   * then that proxy is returned, otherwise a WebServiceException
259   * is thrown.
260   * <p>
261   * Calling this method has the same behavior as the following
262   * <pre>
263   * <code>port = service.getPort(portName, serviceEndpointInterface);</code>
264   * </pre>
265   * where the portName is retrieved from the
266   * metadata of the endpointReference or from the
267   * serviceEndpointInterface and the WSDL
268   * associated with this Service instance.
269   *
270   * @param endpointReference The EndpointReference
271   * for the target service endpoint that will be invoked by the
272   * returned proxy.
273   * @param serviceEndpointInterface Service endpoint interface.
274   * @param features A list of WebServiceFeatures to configure on the
275   * proxy. Supported features not in the features
276   * <code> parameter will have their default values.
277   * @return Object Proxy instance that supports the
278   * specified service endpoint interface.
279   * @throws WebServiceException
280   * <ul>
281   * <li>If there is an error during creation
282   * of the proxy.

```

```

283      *      <LI>If there is any missing WSDL metadata
284      *      as required by this method.
285      *      <LI>If the <code>endpointReference</code> metadata does
286      *      not match the <code>serviceName</code> of this
287      *      <code>Service</code> instance.
288      *      <LI>If a <code>portName</code> cannot be extracted
289      *      from the WSDL or <code>endpointReference</code> metadata.
290      *      <LI>If an invalid
291      *      <code>endpointReference</code>
292      *      is specified.
293      *      <LI>If an invalid
294      *      <code>serviceEndpointInterface</code>
295      *      is specified.
296      *      <LI>If a feature is enabled that is not compatible
297      *      with this port or is unsupported.
298      *      </UL>
299      *
300      * @since JAX-WS 2.1
301      **/
302  public <T> T getPort(EndpointReference endpointReference,
303      Class<T> serviceEndpointInterface, WebServiceFeature... features) {
304      return delegate.getPort(endpointReference, serviceEndpointInterface, features);
305  }
306
307  /**
308   * Creates a new port for the service. Ports created in this way contain
309   * no WSDL port type information and can only be used for creating
310   * <code>Dispatch</code>instances.
311   *
312   * @param portName Qualified name for the target service endpoint.
313   * @param bindingId A String identifier of a binding.
314   * @param endpointAddress Address of the target service endpoint as a URI.
315   * @throws WebServiceException If any error in the creation of
316   * the port.
317   *
318   * @see javax.xml.ws.soap.SOAPBinding#SOAP11HTTP_BINDING
319   * @see javax.xml.ws.soap.SOAPBinding#SOAP12HTTP_BINDING
320   * @see javax.xml.ws.http.HTTPBinding#HTTP_BINDING
321   **/
322  public void addPort(QName portName, String bindingId, String endpointAddress) {
323      delegate.addPort(portName, bindingId, endpointAddress);
324  }
325
326
327  /**
328   * Creates a <code>Dispatch</code> instance for use with objects of
329   * the client's choosing.
330   *
331   * @param portName Qualified name for the target service endpoint
332   * @param type The class of object used for messages or message
333   * payloads. Implementations are required to support
334   * <code>javax.xml.transform.Source</code>, <code>javax.xml.soap.SOAPMessage</code>
335   * and <code>javax.activation.DataSource</code>, depending on

```



```

336     * the binding in use.
337     * @param mode Controls whether the created dispatch instance is message
338     * or payload oriented, i.e. whether the client will work with complete
339     * protocol messages or message payloads. E.g. when using the SOAP
340     * protocol, this parameter controls whether the client will work with
341     * SOAP messages or the contents of a SOAP body. Mode MUST be MESSAGE
342     * when type is SOAPMessage.
343     *
344     * @return Dispatch instance.
345     * @throws WebServiceException If any error in the creation of
346     *         the <code>Dispatch</code> object.
347     *
348     * @see javax.xml.transform.Source
349     * @see javax.xml.soap.SOAPMessage
350     **/
351     public <T> Dispatch<T> createDispatch(QName portName, Class<T> type, Mode mode) {
352         return delegate.createDispatch(portName, type, mode);
353     }
354
355
356     /**
357     * Creates a <code>Dispatch</code> instance for use with objects of
358     * the client's choosing.
359     *
360     * @param portName Qualified name for the target service endpoint
361     * @param type The class of object used for messages or message
362     * payloads. Implementations are required to support
363     * <code>javax.xml.transform.Source</code> and <code>javax.xml.soap.SOAPMessage</code>.
364     * @param mode Controls whether the created dispatch instance is message
365     * or payload oriented, i.e. whether the client will work with complete
366     * protocol messages or message payloads. E.g. when using the SOAP
367     * protocol, this parameter controls whether the client will work with
368     * SOAP messages or the contents of a SOAP body. Mode MUST be <code>MESSAGE</code>
369     * when type is <code>SOAPMessage</code>.
370     * @param features A list of <code>WebServiceFeatures</code> to configure on the
371     * proxy. Supported features not in the <code>features
372     * </code> parameter will have their default values.
373     *
374     * @return Dispatch instance.
375     * @throws WebServiceException If any error in the creation of
376     *         the <code>Dispatch</code> object or if a
377     *         feature is enabled that is not compatible with
378     *         this port or is unsupported.
379     *
380     * @see javax.xml.transform.Source
381     * @see javax.xml.soap.SOAPMessage
382     * @see WebServiceFeature
383     *
384     * @since JAX-WS 2.1
385     **/
386     public <T> Dispatch<T> createDispatch(QName portName, Class<T> type,
387         Service.Mode mode, WebServiceFeature... features) {
388         return delegate.createDispatch(portName, type, mode, features);

```

```

389     }
390
391
392     /**
393      * Creates a Dispatch instance for use with objects of
394      * the client's choosing. If there
395      * are any reference parameters in the
396      * endpointReference, then those reference
397      * parameters MUST appear as SOAP headers, indicating them to be
398      * reference parameters, on all messages sent to the endpoint.
399      * The endpointReference's address MUST be used
400      * for invocations on the endpoint.
401      * In the implementation of this method, the JAX-WS
402      * runtime system takes the responsibility of selecting a protocol
403      * binding (and a port) and configuring the dispatch accordingly from
404      * the WSDL associated with this Service instance or
405      * from the metadata from the endpointReference.
406      * If this Service instance has a WSDL and
407      * the endpointReference
408      * also has a WSDL in its metadata, then the WSDL from this instance MUST be used.
409      * If this Service instance does not have a WSDL and
410      * the endpointReference does have a WSDL, then the
411      * WSDL from the endpointReference MAY be used.
412      * An implementation MUST be able to retrieve the portName from the
413      * endpointReference metadata.
414      * <p>
415      * This method behaves the same as calling
416      * <pre>
417      * dispatch = service.createDispatch(portName, type, mode, features);</code>
418      * </pre>
419      * where the portName is retrieved from the
420      * WSDL or EndpointReference metadata.
421      *
422      * @param endpointReference The EndpointReference
423      * for the target service endpoint that will be invoked by the
424      * returned Dispatch object.
425      * @param type The class of object used to messages or message
426      * payloads. Implementations are required to support
427      * javax.xml.transform.Source and javax.xml.soap.SOAPMessage.
428      * @param mode Controls whether the created dispatch instance is message
429      * or payload oriented, i.e. whether the client will work with complete
430      * protocol messages or message payloads. E.g. when using the SOAP
431      * protocol, this parameter controls whether the client will work with
432      * SOAP messages or the contents of a SOAP body. Mode MUST be MESSAGE
433      * when type is SOAPMessage.
434      * @param features An array of WebServiceFeatures to configure on the
435      * proxy. Supported features not in the features
436      * </code> parameter will have their default values.
437      *
438      * @return Dispatch instance
439      * @throws WebServiceException
440      *
441      * <UL>
442      * <LI>If there is any missing WSDL metadata

```

```

442     *          as required by this method.
443     *          <li>If the <code>endpointReference</code> metadata does
444     *          not match the <code>serviceName</code> or <code>portName</code>
445     *          of a WSDL associated
446     *          with this <code>Service</code> instance.
447     *          <li>If the <code>portName</code> cannot be determined
448     *          from the <code>EndpointReference</code> metadata.
449     *          <li>If any error in the creation of
450     *          the <code>Dispatch</code> object.
451     *          <li>If a feature is enabled that is not
452     *          compatible with this port or is unsupported.
453     *          </UL>
454     *
455     * @see javax.xml.transform.Source
456     * @see javax.xml.soap.SOAPMessage
457     * @see WebServiceFeature
458     *
459     * @since JAX-WS 2.1
460     **/
461     public <T> Dispatch<T> createDispatch(EndpointReference endpointReference,
462         Class<T> type, Service.Mode mode,
463         WebServiceFeature... features) {
464         return delegate.createDispatch(endpointReference, type, mode, features);
465     }
466
467     /**
468     * Creates a <code>Dispatch</code> instance for use with JAXB
469     * generated objects.
470     *
471     * @param portName Qualified name for the target service endpoint
472     * @param context The JAXB context used to marshall and unmarshall
473     * messages or message payloads.
474     * @param mode Controls whether the created dispatch instance is message
475     * or payload oriented, i.e. whether the client will work with complete
476     * protocol messages or message payloads. E.g. when using the SOAP
477     * protocol, this parameter controls whether the client will work with
478     * SOAP messages or the contents of a SOAP body.
479     *
480     * @return Dispatch instance.
481     * @throws WebServiceException If any error in the creation of
482     * the <code>Dispatch</code> object.
483     *
484     * @see javax.xml.bind.JAXBContext
485     **/
486     public Dispatch<Object> createDispatch(QName portName, JAXBContext context,
487         Mode mode) {
488         return delegate.createDispatch(portName, context, mode);
489     }
490
491
492     /**
493     * Creates a <code>Dispatch</code> instance for use with JAXB
494     * generated objects.

```

```

495 *
496 * @param portName Qualified name for the target service endpoint
497 * @param context The JAXB context used to marshal and unmarshal
498 * messages or message payloads.
499 * @param mode Controls whether the created dispatch instance is message
500 * or payload oriented, i.e. whether the client will work with complete
501 * protocol messages or message payloads. E.g. when using the SOAP
502 * protocol, this parameter controls whether the client will work with
503 * SOAP messages or the contents of a SOAP body.
504 * @param features A list of <code>WebServiceFeatures</code> to configure on the
505 * proxy. Supported features not in the <code>features
506 * </code> parameter will have their default values.
507 *
508 * @return Dispatch instance.
509 * @throws WebServiceException If any error in the creation of
510 * the <code>Dispatch</code> object or if a
511 * feature is enabled that is not compatible with
512 * this port or is unsupported.
513 *
514 * @see javax.xml.bind.JAXBContext
515 * @see WebServiceFeature
516 *
517 * @since JAX-WS 2.1
518 */
519 public Dispatch<Object> createDispatch(QName portName,
520     JAXBContext context, Service.Mode mode, WebServiceFeature... features) {
521     return delegate.createDispatch(portName, context, mode, features);
522 }
523
524
525 /**
526 * Creates a <code>Dispatch</code> instance for use with JAXB
527 * generated objects. If there
528 * are any reference parameters in the
529 * <code>endpointReference</code>, then those reference
530 * parameters MUST appear as SOAP headers, indicating them to be
531 * reference parameters, on all messages sent to the endpoint.
532 * The <code>endpointReference's</code> address MUST be used
533 * for invocations on the endpoint.
534 * In the implementation of this method, the JAX-WS
535 * runtime system takes the responsibility of selecting a protocol
536 * binding (and a port) and configuring the dispatch accordingly from
537 * the WSDL associated with this <code>Service</code> instance or
538 * from the metadata from the <code>endpointReference</code>.
539 * If this <code>Service</code> instance has a WSDL and
540 * the <code>endpointReference</code>
541 * also has a WSDL in its metadata, then the WSDL from this instance
542 * MUST be used.
543 * If this <code>Service</code> instance does not have a WSDL and
544 * the <code>endpointReference</code> does have a WSDL, then the
545 * WSDL from the <code>endpointReference</code> MAY be used.
546 * An implementation MUST be able to retrieve the <code>portName</code> from the
547 * <code>endpointReference</code> metadata.

```

```

548 * <p>
549 * This method behaves the same as calling
550 * <pre>
551 * <code>dispatch = service.createDispatch(portName, context, mode, features);</code>
552 * </pre>
553 * where the <code>portName</code> is retrieved from the
554 * WSDL or <code>endpointReference</code> metadata.
555 *
556 * @param endpointReference The <code>EndpointReference</code>
557 * for the target service endpoint that will be invoked by the
558 * returned <code>Dispatch</code> object.
559 * @param context The JAXB context used to marshall and unmarshall
560 * messages or message payloads.
561 * @param mode Controls whether the created dispatch instance is message
562 * or payload oriented, i.e. whether the client will work with complete
563 * protocol messages or message payloads. E.g. when using the SOAP
564 * protocol, this parameter controls whether the client will work with
565 * SOAP messages or the contents of a SOAP body.
566 * @param features An array of <code>WebServiceFeatures</code> to configure on the
567 * proxy. Supported features not in the <code>features
568 * </code> parameter will have their default values.
569 *
570 * @return Dispatch instance
571 * @throws WebServiceException
572 *
573 * <ul>
574 * <li>If there is any missing WSDL metadata
575 * as required by this method.
576 * <li>If the <code>endpointReference</code> metadata does
577 * not match the <code>serviceName</code> or <code>portName</code>
578 * of a WSDL associated
579 * with this <code>Service</code> instance.
580 * <li>If the <code>portName</code> cannot be determined
581 * from the <code>EndpointReference</code> metadata.
582 * <li>If any error in the creation of
583 * the <code>Dispatch</code> object.
584 * <li>if a feature is enabled that is not
585 * compatible with this port or is unsupported.
586 * </ul>
587 *
588 * @see javax.xml.bind.JAXBContext
589 * @see WebServiceFeature
590 *
591 * @since JAX-WS 2.1
592 */
593 public Dispatch<Object> createDispatch(EndpointReference endpointReference,
594     JAXBContext context, Service.Mode mode,
595     WebServiceFeature... features) {
596     return delegate.createDispatch(endpointReference, context, mode, features);
597 }
598 /**
599 * Gets the name of this service.
600 * @return Qualified name of this service

```

```
601     */
602     public QName getServiceName() {
603         return delegate.getServiceName();
604     }
605
606     /**
607      * Returns an Iterator for the list of
608      * QNames of service endpoints grouped by this
609      * service
610      *
611      * @return Returns java.util.Iterator with elements
612      *         of type javax.xml.namespace.QName.
613      * @throws WebServiceException If this Service class does not
614      *         have access to the required WSDL metadata.
615     */
616     public Iterator<javax.xml.namespace.QName> getPorts() {
617         return delegate.getPorts();
618     }
619
620     /**
621      * Gets the location of the WSDL document for this Service.
622      *
623      * @return URL for the location of the WSDL document for
624      *         this service.
625     */
626     public java.net.URL getWSDLDocumentLocation() {
627         return delegate.getWSDLDocumentLocation();
628     }
629
630     /**
631      * Returns the configured handler resolver.
632      *
633      * @return HandlerResolver The HandlerResolver being
634      *         used by this Service instance, or null
635      *         if there isn't one.
636     */
637     public HandlerResolver getHandlerResolver() {
638         return delegate.getHandlerResolver();
639     }
640
641     /**
642      * Sets the HandlerResolver for this Service
643      * instance.
644      *
645      * <p>
646      * The handler resolver, if present, will be called once for each
647      * proxy or dispatch instance that is created, and the handler chain
648      * returned by the resolver will be set on the instance.
649      *
650      * @param handlerResolver The HandlerResolver to use
651      *         for all subsequently created proxy/dispatch objects.
652      *
653      * @see javax.xml.ws.handler.HandlerResolver
654     */
```

```
654 public void setHandlerResolver(HandlerResolver handlerResolver) {
655     delegate.setHandlerResolver(handlerResolver);
656 }
657
658 /**
659  * Returns the executor for this <code>Service</code> instance.
660  *
661  * The executor is used for all asynchronous invocations that
662  * require callbacks.
663  *
664  * @return The <code>java.util.concurrent.Executor</code> to be
665  *         used to invoke a callback.
666  *
667  * @see java.util.concurrent.Executor
668  */
669 public java.util.concurrent.Executor getExecutor() {
670     return delegate.getExecutor();
671 }
672
673 /**
674  * Sets the executor for this <code>Service</code> instance.
675  *
676  * The executor is used for all asynchronous invocations that
677  * require callbacks.
678  *
679  * @param executor The <code>java.util.concurrent.Executor</code>
680  *                 to be used to invoke a callback.
681  *
682  * @throws SecurityException If the instance does not support
683  *                           setting an executor for security reasons (e.g. the
684  *                           necessary permissions are missing).
685  *
686  * @see java.util.concurrent.Executor
687  */
688 public void setExecutor(java.util.concurrent.Executor executor) {
689     delegate.setExecutor(executor);
690 }
691
692 /**
693  * Creates a <code>Service</code> instance.
694  *
695  * The specified WSDL document location and service qualified name MUST
696  * uniquely identify a <code>wsdl:service</code> element.
697  *
698  * @param wsdlDocumentLocation <code>URL</code> for the WSDL document location
699  *                             for the service
700  * @param serviceName <code>QName</code> for the service
701  * @throws WebServiceException If any error in creation of the
702  *                             specified service.
703  */
704 public static Service create(
705     java.net.URL wsdlDocumentLocation,
706     QName serviceName) {
```

```
707     return new Service(wsdlDocumentLocation, serviceName);
708 }
709
710 /**
711  * Creates a <code>Service</code> instance. The created instance is
712  * configured with the web service features.
713  *
714  * The specified WSDL document location and service qualified name MUST
715  * uniquely identify a <code>wsdl:service</code> element.
716  *
717  * @param wsdlDocumentLocation <code>URL</code> for the WSDL document location
718  *                               for the service
719  * @param serviceName <code>QName</code> for the service
720  * @param features Web Service features that must be configured on
721  *                 the service. If the provider doesn't understand a feature,
722  *                 it must throw a WebServiceException.
723  * @throws WebServiceException If any error in creation of the
724  *                               specified service.
725  * @since JAX-WS 2.2
726  */
727 public static Service create(
728     java.net.URL wsdlDocumentLocation,
729     QName serviceName, WebServiceFeature ... features) {
730     return new Service(wsdlDocumentLocation, serviceName, features);
731 }
732
733 /**
734  * Creates a <code>Service</code> instance.
735  *
736  * @param serviceName <code>QName</code> for the service
737  * @throws WebServiceException If any error in creation of the
738  *                               specified service
739  */
740 public static Service create(QName serviceName) {
741     return new Service(null, serviceName);
742 }
743
744 /**
745  * Creates a <code>Service</code> instance. The created instance is
746  * configured with the web service features.
747  *
748  * @param serviceName <code>QName</code> for the service
749  * @param features Web Service features that must be configured on
750  *                 the service. If the provider doesn't understand a feature,
751  *                 it must throw a WebServiceException.
752  * @throws WebServiceException If any error in creation of the
753  *                               specified service
754  *
755  * @since JAX-WS 2.2
756  */
757 public static Service create(QName serviceName, WebServiceFeature ... features) {
758     return new Service(null, serviceName, features);
759 }
```



760 }

**27.21 ServiceMode.java**

Secuencia 280: javax/xml/ws/ServiceMode.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.ElementType;
32 import java.lang.annotation.RetentionPolicy;
33 import java.lang.annotation.Inherited;
34
35 /**
36 * Used to indicate whether a {@link Provider} implementation wishes to work
37 * with entire protocol messages or just with protocol message payloads.
38 *
39 * @since JAX-WS 2.0
40 */
41 @Target({ElementType.TYPE})
42 @Retention(RetentionPolicy.RUNTIME)
43 @Inherited
44 @Documented
45 public @interface ServiceMode {
46     /**
47      * Service mode. <code>PAYLOAD</code> indicates that the <code>Provider</code> implementation
48      * wishes to work with protocol message payloads only. <code>MESSAGE</code> indicates
```

```
49  * that the <code>Provider</code> implementation wishes to work with entire protocol
50  * messages.
51  **/
52  public Service.Mode value() default Service.Mode.PAYLOAD;
53 }
```

## 27.22 WebEndpoint.java

Secuencia 281: javax/xml/ws/WebEndpoint.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.ElementType;
32 import java.lang.annotation.RetentionPolicy;
33
34 /**
35  * Used to annotate the <code>get<em>PortName</em>()</code>
36  * methods of a generated service interface.
37  *
38  * <p>The information specified in this annotation is sufficient
39  * to uniquely identify a <code>wsdl:port</code> element
40  * inside a <code>wsdl:service</code>. The latter is
41  * determined based on the value of the <code>WebServiceClient</code>
42  * annotation on the generated service interface itself.
43  *
44  * @since JAX-WS 2.0
```

```

45  *
46  * @see javax.xml.ws.WebServiceClient
47  **/
48  @Target({ElementType.METHOD})
49  @Retention(RetentionPolicy.RUNTIME)
50  @Documented
51  public @interface WebEndpoint {
52      /**
53       * The local name of the endpoint.
54       */
55      String name() default "";
56  }

```

## 27.23 WebFault.java

Secuencia 282: javax/xml/ws/WebFault.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.ElementType;
32 import java.lang.annotation.RetentionPolicy;
33
34 /**
35  * Used to annotate service specific exception classes to customize
36  * to the local and namespace name of the fault element and the name
37  * of the fault bean.

```

```

38  *
39  * @since JAX-WS 2.0
40  **/
41  @Target({ElementType.TYPE})
42  @Retention(RetentionPolicy.RUNTIME)
43  @Documented
44  public @interface WebFault {
45      /**
46       * Element's local name.
47       */
48      public String name() default "";
49
50      /**
51       * Element's namespace name.
52       */
53      public String targetNamespace() default "";
54
55      /**
56       * Fault bean name.
57       */
58      public String faultBean() default "";
59
60
61      /**
62       * wsdl:Message's name. Default name is the exception's class name.
63       * @since JAX-WS 2.2
64       */
65      public String messageName() default "";
66
67  }

```

## 27.24 WebServiceClient.java

Secuencia 283: javax/xml/ws/WebServiceClient.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```

```

20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.ws;
27
28  import java.lang.annotation.Documented;
29  import java.lang.annotation.Target;
30  import java.lang.annotation.Retention;
31  import java.lang.annotation.ElementType;
32  import java.lang.annotation.RetentionPolicy;
33
34  /**
35   * Used to annotate a generated service interface.
36   *
37   * <p>The information specified in this annotation is sufficient
38   * to uniquely identify a <code>wsdl:service</code>
39   * element inside a WSDL document. This <code>wsdl:service</code>
40   * element represents the Web service for which the generated
41   * service interface provides a client view.
42   *
43   * @since JAX-WS 2.0
44  */
45  @Target({ElementType.TYPE})
46  @Retention(RetentionPolicy.RUNTIME)
47  @Documented
48  public @interface WebServiceClient {
49      /**
50       * The local name of the Web service.
51       */
52      String name() default "";
53
54      /**
55       * The namespace for the Web service.
56       */
57      String targetNamespace() default "";
58
59      /**
60       * The location of the WSDL document for the service (a URL).
61       */
62      String wsdlLocation() default "";
63  }

```

## 27.25 WebServiceContext.java

Secuencia 284: javax/xml/ws/WebServiceContext.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *

```

```
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.security.Principal;
29 import javax.xml.ws.handler.MessageContext;
30 import javax.xml.ws.wsaddressing.W3CEndpointReference;
31 import org.w3c.dom.Element;
32
33
34 /**
35  * A WebServiceContext makes it possible for
36  * a web service endpoint implementation class to access
37  * message context and security information relative to
38  * a request being served.
39  *
40  * Typically a WebServiceContext is injected
41  * into an endpoint implementation class using the
42  * Resource annotation.
43  *
44  * @since JAX-WS 2.0
45  *
46  * @see javax.annotation.Resource
47  */
48 public interface WebServiceContext {
49
50     /**
51      * Returns the MessageContext for the request being served
52      * at the time this method is called. Only properties with
53      * APPLICATION scope will be visible to the application.
54      *
55      * @return MessageContext The message context.
56      *
57      * @throws IllegalStateException This exception is thrown
58      *         if the method is called while no request is
```

```

59      *      being serviced.
60      *
61      * @see javax.xml.ws.handler.MessageContext
62      * @see javax.xml.ws.handler.MessageContext.Scope
63      * @see java.lang.IllegalStateException
64      **/
65 public MessageContext getMessageContext();
66
67 /**
68  * Returns the Principal that identifies the sender
69  * of the request currently being serviced. If the
70  * sender has not been authenticated, the method
71  * returns <code>null</code>.
72  *
73  * @return Principal The principal object.
74  *
75  * @throws IllegalStateException This exception is thrown
76  *         if the method is called while no request is
77  *         being serviced.
78  *
79  * @see java.security.Principal
80  * @see java.lang.IllegalStateException
81  **/
82 public Principal getUserPrincipal();
83
84 /**
85  * Returns a boolean indicating whether the
86  * authenticated user is included in the specified
87  * logical role. If the user has not been
88  * authenticated, the method returns <code>false</code>.
89  *
90  * @param role A <code>String</code> specifying the name of the role
91  *
92  * @return a <code>boolean</code> indicating whether
93  * the sender of the request belongs to a given role
94  *
95  * @throws IllegalStateException This exception is thrown
96  *         if the method is called while no request is
97  *         being serviced.
98  **/
99 public boolean isUserInRole(String role);
100
101 /**
102  * Returns the <code>EndpointReference</code> for this
103  * endpoint.
104  * <p>
105  * If the {@link Binding} for this <code>bindingProvider</code> is
106  * either SOAP1.1/HTTP or SOAP1.2/HTTP, then a
107  * <code>W3CEndpointReference</code> MUST be returned.
108  *
109  * @param referenceParameters Reference parameters to be associated with the
110  * returned <code>EndpointReference</code> instance.
111  * @return EndpointReference of the endpoint associated with this

```

```

112 * <code>WebServiceContext</code>.
113 * If the returned <code>EndpointReference</code> is of type
114 * <code>W3CEndpointReference</code> then it MUST contain the
115 * the specified <code>referenceParameters</code>.
116 *
117 * @throws IllegalStateException This exception is thrown
118 *         if the method is called while no request is
119 *         being serviced.
120 *
121 * @see W3CEndpointReference
122 *
123 * @since JAX-WS 2.1
124 */
125 public EndpointReference getEndpointReference(Element... referenceParameters);
126
127 /**
128 * Returns the <code>EndpointReference</code> associated with
129 * this endpoint.
130 *
131 * @param clazz The type of <code>EndpointReference</code> that
132 * MUST be returned.
133 * @param referenceParameters Reference parameters to be associated with the
134 * returned <code>EndpointReference</code> instance.
135 * @return EndpointReference of type <code>clazz</code> of the endpoint
136 * associated with this <code>WebServiceContext</code> instance.
137 * If the returned <code>EndpointReference</code> is of type
138 * <code>W3CEndpointReference</code> then it MUST contain the
139 * the specified <code>referenceParameters</code>.
140 *
141 * @throws IllegalStateException This exception is thrown
142 *         if the method is called while no request is
143 *         being serviced.
144 * @throws WebServiceException If the <code>clazz</code> type of
145 * <code>EndpointReference</code> is not supported.
146 *
147 * @since JAX-WS 2.1
148 */
149 public <T extends EndpointReference> T getEndpointReference(Class<T> clazz,
150         Element... referenceParameters);
151 }

```

## 27.26 WebServiceException.java

Secuencia 285: javax/xml/ws/WebServiceException.java

```

1 /*
2 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4 *
5 *
6 *
7 *
8 *
9 *

```



```
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws;
27
28 /** The <code>WebServiceException</code> class is the base
29  * exception class for all JAX-WS API runtime exceptions.
30  *
31  * @since JAX-WS 2.0
32  */
33
34 public class WebServiceException extends java.lang.RuntimeException {
35
36     /** Constructs a new exception with <code>null</code> as its
37      * detail message. The cause is not initialized.
38      */
39     public WebServiceException() {
40         super();
41     }
42
43     /** Constructs a new exception with the specified detail
44      * message. The cause is not initialized.
45      * @param message The detail message which is later
46      *               retrieved using the getMessage method
47      */
48     public WebServiceException(String message) {
49         super(message);
50     }
51
52     /** Constructs a new exception with the specified detail
53      * message and cause.
54      *
55      * @param message The detail message which is later retrieved
56      *               using the getMessage method
57      * @param cause    The cause which is saved for the later
58      *               retrieval throw by the getCause method
59      */
60     public WebServiceException(String message, Throwable cause) {
61         super(message, cause);
62     }
63 }
```

```

63
64 /** Constructs a new WebServiceException with the specified cause
65  * and a detail message of <tt>(cause==null ? null :
66  * cause.toString())</tt> (which typically contains the
67  * class and detail message of <tt>cause</tt>).
68  *
69  * @param cause    The cause which is saved for the later
70  *                  retrieval throw by the getCause method.
71  *                  (A <tt>null</tt> value is permitted, and
72  *                  indicates that the cause is nonexistent or
73  *                  unknown.)
74  */
75 public WebServiceException(Throwable cause) {
76     super(cause);
77 }
78
79 }

```

## 27.27 WebServiceFeature.java

Secuencia 286: javax/xml/ws/WebServiceFeature.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28
29 /**
30  * A WebServiceFeature is used to represent a feature that can be
31  * enabled or disabled for a web service.
32  * <p>

```

```

33  * The JAX-WS specification will define some standard features and
34  * JAX-WS implementors are free to define additional features if
35  * necessary. Vendor specific features may not be portable so
36  * caution should be used when using them. Each Feature definition
37  * MUST define a <code>public static final String ID</code>
38  * that can be used in the Feature annotation to refer
39  * to the feature. This ID MUST be unique across all features
40  * of all vendors. When defining a vendor specific feature ID,
41  * use a vendor specific namespace in the ID string.
42  *
43  * @see javax.xml.ws.RespectBindingFeature
44  * @see javax.xml.ws.soap.AddressingFeature
45  * @see javax.xml.ws.soap.MTOMFeature
46  *
47  * @since 2.1
48  */
49  public abstract class WebServiceFeature {
50      /**
51       * Each Feature definition MUST define a public static final
52       * String ID that can be used in the Feature annotation to refer
53       * to the feature.
54       */
55      // public static final String ID = "some unique feature Identifier";
56
57      /**
58       * Get the unique identifier for this WebServiceFeature.
59       *
60       * @return the unique identifier for this feature.
61       */
62      public abstract String getID();
63
64      /**
65       * Specifies if the feature is enabled or disabled
66       */
67      protected boolean enabled = false;
68
69
70      protected WebServiceFeature(){}
71
72
73      /**
74       * Returns <code>true</code> if this feature is enabled.
75       *
76       * @return <code>true</code> if and only if the feature is enabled .
77       */
78      public boolean isEnabled() {
79          return enabled;
80      }
81  }

```

## 27.28 WebServicePermission.java

Secuencia 287: javax/xml/ws/WebServicePermission.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.security.BasicPermission;
29
30 /**
31 * This class defines web service permissions.
32 * <p>
33 * Web service Permissions are identified by name (also referred to as
34 * a "target name") alone. There are no actions associated
35 * with them.
36 * <p>
37 * The following permission target name is defined:
38 * <p>
39 * <dl>
40 *   <dt>publishEndpoint
41 * </dt>
42 * <p>
43 * The <code>publishEndpoint</code> permission allows publishing a
44 * web service endpoint using the <code>publish</code> methods
45 * defined by the <code>javax.xml.ws.Endpoint</code> class.
46 * <p>
47 * Granting <code>publishEndpoint</code> allows the application to be
48 * exposed as a network service. Depending on the security of the runtime and
49 * the security of the application, this may introduce a security hole that
50 * is remotely exploitable.
51 *
52 * @see javax.xml.ws.Endpoint
53 * @see java.security.BasicPermission
```

```

54  * @see java.security.Permission
55  * @see java.security.Permissions
56  * @see java.lang.SecurityManager
57  * @see java.net.SocketPermission
58  */
59  public final class WebServicePermission extends BasicPermission {
60
61      private static final long serialVersionUID = -146474640053770988L;
62
63      /**
64       * Creates a new permission with the specified name.
65       *
66       * @param name the name of the <code>WebServicePermission</code>
67       */
68      public WebServicePermission(String name) {
69          super(name);
70      }
71
72      /**
73       * Creates a new permission with the specified name and actions.
74       *
75       * The <code>actions</code> parameter is currently unused and
76       * it should be <code>null</code>.
77       *
78       * @param name the name of the <code>WebServicePermission</code>
79       * @param actions should be <code>null</code>
80       */
81      public WebServicePermission(String name, String actions) {
82          super(name, actions);
83      }
84
85  }

```

## 27.29 WebServiceProvider.java

Secuencia 288: javax/xml/ws/WebServiceProvider.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```

18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import java.lang.annotation.ElementType;
32 import java.lang.annotation.RetentionPolicy;
33 /**
34  * Used to annotate a Provider implementation class.
35  *
36  * @since JAX-WS 2.0
37  * @see javax.xml.ws.Provider
38  */
39 @Target(ElementType.TYPE)
40 @Retention(RetentionPolicy.RUNTIME)
41 @Documented
42 public @interface WebServiceProvider {
43     /**
44      * Location of the WSDL description for the service.
45      */
46     String wsdlLocation() default "";
47
48     /**
49      * Service name.
50      */
51     String serviceName() default "";
52
53     /**
54      * Target namespace for the service
55      */
56     String targetNamespace() default "";
57
58     /**
59      * Port name.
60      */
61     String portName() default "";
62 }

```

## 27.30 WebServiceRef.java

Secuencia 289: javax/xml/ws/WebServiceRef.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *

```

```
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import javax.xml.ws.soap.Addressing;
29 import javax.xml.ws.spi.WebServiceFeatureAnnotation;
30 import java.lang.annotation.Documented;
31 import java.lang.annotation.Target;
32 import java.lang.annotation.ElementType;
33 import java.lang.annotation.Retention;
34 import java.lang.annotation.RetentionPolicy;
35
36 /**
37  * The <code>WebServiceRef</code> annotation is used to
38  * define a reference to a web service and
39  * (optionally) an injection target for it.
40  * It can be used to inject both service and proxy
41  * instances. These injected references are not thread safe.
42  * If the references are accessed by multiple threads,
43  * usual synchronization techniques can be used to
44  * support multiple threads.
45  *
46  * <p>
47  * Web service references are resources in the Java EE 5 sense.
48  * The annotations (for example, {@link Addressing}) annotated with
49  * meta-annotation {@link WebServiceFeatureAnnotation}
50  * can be used in conjunction with <code>WebServiceRef</code>.
51  * The created reference MUST be configured with annotation's web service
52  * feature.
53  *
54  * <p>
55  * For example, in the code below, the injected
56  * <code>StockQuoteProvider</code> proxy MUST
57  * have WS-Addressing enabled as specified by the
```

```

58 * {@link Addressing}
59 * annotation.
60 *
61 * <pre><code>
62 *     public class MyClient {
63 *         &#64;Addressing
64 *         &#64;WebServiceRef(StockQuoteService.class)
65 *         private StockQuoteProvider stockQuoteProvider;
66 *         ...
67 *     }
68 * </code></pre>
69 *
70 * <p>
71 * If a JAX-WS implementation encounters an unsupported or unrecognized
72 * annotation annotated with the <code>WebServiceFeatureAnnotation</code>
73 * that is specified with <code>WebServiceRef</code>, an ERROR MUST be given.
74 *
75 * @see javax.annotation.Resource
76 * @see WebServiceFeatureAnnotation
77 *
78 * @since JAX-WS 2.0
79 *
80 **/
81
82 @Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
83 @Retention(RetentionPolicy.RUNTIME)
84 @Documented
85 public @interface WebServiceRef {
86     /**
87      * The JNDI name of the resource. For field annotations,
88      * the default is the field name. For method annotations,
89      * the default is the JavaBeans property name corresponding
90      * to the method. For class annotations, there is no default
91      * and this MUST be specified.
92      *
93      * The JNDI name can be absolute(with any logical namespace) or relative
94      * to JNDI <code>java:comp/env</code> namespace.
95      */
96     String name() default "";
97
98     /**
99      * The Java type of the resource. For field annotations,
100      * the default is the type of the field. For method annotations,
101      * the default is the type of the JavaBeans property.
102      * For class annotations, there is no default and this MUST be
103      * specified.
104      */
105     Class<?> type() default Object.class;
106
107     /**
108      * A product specific name that this resource should be mapped to.
109      * The name of this resource, as defined by the <code>name</code>
110      * element or defaulted, is a name that is local to the application

```



```

111     * component using the resource. (When a relative JNDI name
112     * is specified, then it's a name in the JNDI
113     * <code>java:comp/env</code> namespace.) Many application servers
114     * provide a way to map these local names to names of resources
115     * known to the application server. This mapped name is often a
116     * <i>global</i> JNDI name, but may be a name of any form.
117     * <p>
118     * Application servers are not required to support any particular
119     * form or type of mapped name, nor the ability to use mapped names.
120     * The mapped name is product-dependent and often installation-dependent.
121     * No use of a mapped name is portable.
122     */
123     String mappedName() default "";
124
125     /**
126     * The service class, always a type extending
127     * <code>javax.xml.ws.Service</code>. This element MUST be specified
128     * whenever the type of the reference is a service endpoint interface.
129     */
130     // 2.1 has Class value() default Object.class;
131     // Fixing this raw Class type correctly in 2.2 API. This shouldn't cause
132     // any compatibility issues for applications.
133     Class<? extends Service> value() default Service.class;
134
135     /**
136     * A URL pointing to the WSDL document for the web service.
137     * If not specified, the WSDL location specified by annotations
138     * on the resource type is used instead.
139     */
140     String wsdlLocation() default "";
141
142     /**
143     * A portable JNDI lookup name that resolves to the target
144     * web service reference.
145     *
146     * @since JAX-WS 2.2
147     */
148     String lookup() default "";
149
150 }

```

## 27.31 WebServiceRefs.java

Secuencia 290: javax/xml/ws/WebServiceRefs.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```

```
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.Retention;
31 import static java.lang.annotation.ElementType.*;
32 import static java.lang.annotation.RetentionPolicy.*;
33
34 /**
35  * The WebServiceRefs annotation allows
36  * multiple web service references to be declared at the
37  * class level.
38  *
39  * <p>
40  * It can be used to inject both service and proxy
41  * instances. These injected references are not thread safe.
42  * If the references are accessed by multiple threads,
43  * usual synchronization techniques can be used to
44  * support multiple threads.
45  *
46  * <p>
47  * There is no way to associate web service features with
48  * the injected instances. If an instance needs to be
49  * configured with web service features, use @WebServiceRef
50  * to inject the resource along with its features.
51  *
52  * <p>
53  * <b>Example</b>: The StockQuoteProvider
54  * proxy instance, and the StockQuoteService service
55  * instance are injected using @WebServiceRefs.
56  *
57  * <pre><code>
58  *     &#64;WebServiceRefs({&#64;WebServiceRef(name="service/stockquoteservice", value=
59  *         StockQuoteService.class),
60  *         &#64;WebServiceRef(name="service/stockquoteprovider", type=
61  *             StockQuoteProvider.class, value=StockQuoteService.class)})
62  *     public class MyClient {
```

```

61 *     void init() {
62 *         Context ic = new InitialContext();
63 *         StockQuoteService service = (StockQuoteService) ic.lookup("java:comp/env/service/
stockquoteservice");
64 *         StockQuoteProvider port = (StockQuoteProvider) ic.lookup("java:comp/env/service/
stockquoteprovider");
65 *         ...
66 *     }
67 *     ...
68 * }
69 * </code></pre>
70 *
71 * @see WebServiceRef
72 * @since 2.0
73 */
74
75 @Documented
76 @Retention(RUNTIME)
77 @Target(TYPE)
78 public @interface WebServiceRefs {
79     /**
80      * Array used for multiple web service reference declarations.
81      */
82     WebServiceRef[] value();
83 }

```

## 28 Xml Ws Handler (javax.xml.ws.handler package)

### 28.1 Handler.java

Secuencia 291: javax/xml/ws/handler/Handler.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *

```

```
23  *
24  */
25
26 package javax.xml.ws.handler;
27
28 import javax.xml.ws.ProtocolException;
29 import javax.xml.ws.handler.MessageContext;
30
31 /** The Handler interface
32  * is the base interface for JAX-WS handlers.
33  *
34  * @since JAX-WS 2.0
35  */
36 public interface Handler<C extends MessageContext> {
37
38     /** The handleMessage method is invoked for normal processing
39     * of inbound and outbound messages. Refer to the description of the handler
40     * framework in the JAX-WS specification for full details.
41     *
42     * @param context the message context.
43     * @return An indication of whether handler processing should continue for
44     * the current message
45     *
46     * 


47     * - Return true to continue
48     * processing.

49     * - Return false to block
50     * processing.

51     * 

52     *
53     * @throws RuntimeException Causes the JAX-WS runtime to cease
54     * handler processing and generate a fault.
55     * @throws ProtocolException Causes the JAX-WS runtime to switch to
56     * fault message processing.
57     */
58     public boolean handleMessage(C context);
59
60     /** The handleFault method is invoked for fault message
61     * processing. Refer to the description of the handler
62     * framework in the JAX-WS specification for full details.
63     *
64     * @param context the message context
65     * @return An indication of whether handler fault processing should continue
66     * for the current message
67     *
68     * 


69     * - Return true to continue
70     * processing.

71     * - Return false to block
72     * processing.

73     * 

74     *
75     * @throws RuntimeException Causes the JAX-WS runtime to cease
76     * handler fault processing and dispatch the fault.
77     * @throws ProtocolException Causes the JAX-WS runtime to cease
78     * handler fault processing and dispatch the fault.
79     */
80 }
```

```

76 public boolean handleFault(C context);
77
78 /**
79  * Called at the conclusion of a message exchange pattern just prior to
80  * the JAX-WS runtime dispatching a message, fault or exception. Refer to
81  * the description of the handler
82  * framework in the JAX-WS specification for full details.
83  *
84  * @param context the message context
85  */
86 public void close(MessageContext context);
87 }

```

## 28.2 HandlerResolver.java

Secuencia 292: javax/xml/ws/handler/HandlerResolver.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.handler;
27
28 /**
29  * <code>HandlerResolver</code> is an interface implemented
30  * by an application to get control over the handler chain
31  * set on proxy/dispatch objects at the time of their creation.
32  * <p>
33  * A <code>HandlerResolver</code> may be set on a <code>Service</code>
34  * using the <code>setHandlerResolver</code> method.
35  * <p>
36  * When the runtime invokes a <code>HandlerResolver</code>, it will
37  * pass it a <code>PortInfo</code> object containing information

```

```

38  * about the port that the proxy/dispatch object will be accessing.
39  *
40  * @see javax.xml.ws.Service#setHandlerResolver
41  *
42  * @since JAX-WS 2.0
43  **/
44  public interface HandlerResolver {
45
46      /**
47       * Gets the handler chain for the specified port.
48       *
49       * @param portInfo Contains information about the port being accessed.
50       * @return java.util.List<Handler> chain
51       */
52      public java.util.List<Handler> getHandlerChain(PortInfo portInfo);
53  }

```

### 28.3 LogicalHandler.java

Secuencia 293: javax/xml/ws/handler/LogicalHandler.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.handler;
27
28 /** The <code>LogicalHandler</code> extends
29  * Handler to provide typesafety for the message context parameter.
30  *
31  * @since JAX-WS 2.0
32  **/
33 public interface LogicalHandler<C extends LogicalMessageContext> extends Handler<C> {

```

34 }

## 28.4 LogicalMessageContext.java

Secuencia 294: javax/xml/ws/handler/LogicalMessageContext.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.handler;
27
28 import javax.xml.ws.LogicalMessage;
29
30 /** The LogicalMessageContext interface extends
31 * MessageContext to
32 * provide access to a the contained message as a protocol neutral
33 * LogicalMessage
34 *
35 * @since JAX-WS 2.0
36 */
37 public interface LogicalMessageContext
38     extends MessageContext {
39
40     /** Gets the message from this message context
41     *
42     * @return The contained message; returns null if no
43     *         message is present in this message context
44     */
45     public LogicalMessage getMessage();
46 }
```

## 28.5 MessageContext.java

Secuencia 295: javax/xml/ws/handler/MessageContext.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.handler;
27 import java.util.Map;
28
29 /**
30 * The interface MessageContext abstracts the message
31 * context that is processed by a handler in the handle
32 * method.
33 *
34 * <p>The MessageContext interface provides methods to
35 * manage a property set. MessageContext properties
36 * enable handlers in a handler chain to share processing related
37 * state.
38 *
39 * @since JAX-WS 2.0
40 */
41 public interface MessageContext extends Map<String, Object> {
42
43     /**
44      * Standard property: message direction, true for
45      * outbound messages, false for inbound.
46      * <p>Type: boolean
47      */
48     public static final String MESSAGE_OUTBOUND_PROPERTY =
49         "javax.xml.ws.handler.message.outbound";
50
51     /**
52      * Standard property: Map of attachments to a message for the inbound
```



```
53     * message, key is the MIME Content-ID, value is a DataHandler.
54     * <p>Type: java.util.Map<String,DataHandler>
55     */
56     public static final String INBOUND_MESSAGE_ATTACHMENTS =
57         "javax.xml.ws.binding.attachments.inbound";
58
59     /**
60     * Standard property: Map of attachments to a message for the outbound
61     * message, key is the MIME Content-ID, value is a DataHandler.
62     * <p>Type: java.util.Map<String,DataHandler>
63     */
64     public static final String OUTBOUND_MESSAGE_ATTACHMENTS =
65         "javax.xml.ws.binding.attachments.outbound";
66
67     /**
68     * Standard property: input source for WSDL document.
69     * <p>Type: org.xml.sax.InputSource
70     */
71     public static final String WSDL_DESCRIPTION =
72         "javax.xml.ws.wsdl.description";
73
74     /**
75     * Standard property: name of WSDL service.
76     * <p>Type: javax.xml.namespace.QName
77     */
78     public static final String WSDL_SERVICE =
79         "javax.xml.ws.wsdl.service";
80
81     /**
82     * Standard property: name of WSDL port.
83     * <p>Type: javax.xml.namespace.QName
84     */
85     public static final String WSDL_PORT =
86         "javax.xml.ws.wsdl.port";
87
88     /**
89     * Standard property: name of wsdl interface (2.0) or port type (1.1).
90     * <p>Type: javax.xml.namespace.QName
91     */
92     public static final String WSDL_INTERFACE =
93         "javax.xml.ws.wsdl.interface";
94
95     /**
96     * Standard property: name of WSDL operation.
97     * <p>Type: javax.xml.namespace.QName
98     */
99     public static final String WSDL_OPERATION =
100         "javax.xml.ws.wsdl.operation";
101
102     /**
103     * Standard property: HTTP response status code.
104     * <p>Type: java.lang.Integer
105     */
```

```
106 public static final String HTTP_RESPONSE_CODE =
107     "javax.xml.ws.http.response.code";
108
109 /**
110  * Standard property: HTTP request headers.
111  * <p>Type: java.util.Map<java.lang.String, java.util.List<java.lang.String>>
112  */
113 public static final String HTTP_REQUEST_HEADERS =
114     "javax.xml.ws.http.request.headers";
115
116 /**
117  * Standard property: HTTP response headers.
118  * <p>Type: java.util.Map<java.lang.String, java.util.List<java.lang.String>>
119  */
120 public static final String HTTP_RESPONSE_HEADERS =
121     "javax.xml.ws.http.response.headers";
122
123 /**
124  * Standard property: HTTP request method.
125  * <p>Type: java.lang.String
126  */
127 public static final String HTTP_REQUEST_METHOD =
128     "javax.xml.ws.http.request.method";
129
130 /**
131  * Standard property: servlet request object.
132  * <p>Type: javax.servlet.http.HttpServletRequest
133  */
134 public static final String SERVLET_REQUEST =
135     "javax.xml.ws.servlet.request";
136
137 /**
138  * Standard property: servlet response object.
139  * <p>Type: javax.servlet.http.HttpServletResponse
140  */
141 public static final String SERVLET_RESPONSE =
142     "javax.xml.ws.servlet.response";
143
144 /**
145  * Standard property: servlet context object.
146  * <p>Type: javax.servlet.ServletContext
147  */
148 public static final String SERVLET_CONTEXT =
149     "javax.xml.ws.servlet.context";
150
151 /**
152  * Standard property: Query string for request.
153  * <p>Type: String
154  */
155 public static final String QUERY_STRING =
156     "javax.xml.ws.http.request.querystring";
157
158 /**
```

```

159     * Standard property: Request Path Info
160     * <p>Type: String
161     */
162     public static final String PATH_INFO =
163         "javax.xml.ws.http.request.pathinfo";
164
165     /**
166     * Standard property: WS Addressing Reference Parameters.
167     * The list MUST include all SOAP headers marked with the
168     * wsa:IsReferenceParameter="true" attribute.
169     * <p>Type: List<Element>
170     *
171     * @since JAX-WS 2.1
172     */
173     public static final String REFERENCE_PARAMETERS =
174         "javax.xml.ws.reference.parameters";
175
176     /**
177     * Property scope. Properties scoped as <code>APPLICATION</code> are
178     * visible to handlers,
179     * client applications and service endpoints; properties scoped as
180     * <code>HANDLER</code>
181     * are only normally visible to handlers.
182     */
183     public enum Scope {APPLICATION, HANDLER};
184
185     /**
186     * Sets the scope of a property.
187     *
188     * @param name Name of the property associated with the
189     *             <code>MessageContext</code>
190     * @param scope Desired scope of the property
191     * @throws java.lang.IllegalArgumentException if an illegal
192     *         property name is specified
193     */
194     public void setScope(String name, Scope scope);
195
196     /**
197     * Gets the scope of a property.
198     *
199     * @param name Name of the property
200     * @return Scope of the property
201     * @throws java.lang.IllegalArgumentException if a non-existent
202     *         property name is specified
203     */
204     public Scope getScope(String name);
205 }

```

## 28.6 PortInfo.java

Secuencia 296: javax/xml/ws/handler/PortInfo.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.

```

```
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.handler;
27
28 import javax.xml.namespace.QName;
29
30 /**
31  * The <code>PortInfo</code> interface is used by a
32  * <code>HandlerResolver</code> to query information about
33  * the port it is being asked to create a handler chain for.
34  * <p>
35  * This interface is never implemented by an application,
36  * only by a JAX-WS implementation.
37  *
38  * @since JAX-WS 2.0
39  */
40 public interface PortInfo {
41
42     /**
43      * Gets the qualified name of the WSDL service name containing
44      * the port being accessed.
45      *
46      * @return javax.xml.namespace.QName The qualified name of the WSDL service.
47      */
48     public QName getServiceName();
49
50     /**
51      * Gets the qualified name of the WSDL port being accessed.
52      *
53      * @return javax.xml.namespace.QName The qualified name of the WSDL port.
54      */
55     public QName getPortName();
56 }
```

```

56
57 /**
58  * Gets the URI identifying the binding used by the port being accessed.
59  *
60  * @return String The binding identifier for the port.
61  *
62  * @see javax.xml.ws.Binding
63  */
64 public String getBindingID();
65
66 }

```

## 29 Xml Ws Handler Soap (javax.xml.ws.handler.soap package)

### 29.1 SOAPHandler.java

Secuencia 297: javax/xml/ws/handler/soap/SOAPHandler.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.handler.soap;
27
28 import javax.xml.namespace.QName;
29 import javax.xml.ws.handler.Handler;
30 import java.util.Set;
31
32 /** The <code>SOAPHandler</code> class extends <code>Handler</code>
33  * to provide typesafety for the message context parameter and add a method
34  * to obtain access to the headers that may be processed by the handler.
35  *
36  * @since JAX-WS 2.0

```

```

37  /**/
38  public interface SOAPHandler<T extends SOAPMessageContext>
39      extends Handler<T> {
40
41      /** Gets the header blocks that can be processed by this Handler
42       * instance.
43       *
44       * @return Set of <code>QNames</code> of header blocks processed by this
45       * handler instance. <code>QName</code> is the qualified
46       * name of the outermost element of the Header block.
47       */
48      Set<QName> getHeaders();
49  }

```

## 29.2 SOAPMessageContext.java

Secuencia 298: javax/xml/ws/handler/soap/SOAPMessageContext.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.handler.soap;
27
28 import javax.xml.soap.SOAPMessage;
29 import javax.xml.bind.JAXBContext;
30 import javax.xml.namespace.QName;
31 import java.util.Set;
32
33 /** The interface <code>SOAPMessageContext</code>
34  * provides access to the SOAP message for either RPC request or
35  * response. The <code>javax.xml.soap.SOAPMessage</code> specifies
36  * the standard Java API for the representation of a SOAP 1.1 message

```

```
37  * with attachments.
38  *
39  * @see javax.xml.soap.SOAPMessage
40  *
41  * @since JAX-WS 2.0
42  **/
43  public interface SOAPMessageContext
44      extends javax.xml.ws.handler.MessageContext {
45
46      /** Gets the SOAPMessage from this message context. Modifications
47       * to the returned SOAPMessage change the message in-place, there
48       * is no need to subsequently call setMessage.
49       *
50       * @return Returns the SOAPMessage; returns null if no
51       *         SOAPMessage is present in this message context
52       */
53      public SOAPMessage getMessage();
54
55      /** Sets the SOAPMessage in this message context
56       *
57       * @param message SOAP message
58       * @throws WebServiceException If any error during the setting
59       *         of the SOAPMessage in this message context
60       * @throws java.lang.UnsupportedOperationException If this
61       *         operation is not supported
62       */
63      public void setMessage(SOAPMessage message);
64
65      /** Gets headers that have a particular qualified name from the message in the
66       * message context. Note that a SOAP message can contain multiple headers
67       * with the same qualified name.
68       *
69       * @param header The XML qualified name of the SOAP header(s).
70       * @param context The JAXBContext that should be used to unmarshall the
71       *         header
72       * @param allRoles If true then returns headers for all SOAP
73       *         roles, if false then only returns headers targetted
74       *         at the roles currently being played by this SOAP node, see
75       *         getRoles.
76       * @return An array of unmarshalled headers; returns an empty array if no
77       *         message is present in this message context or no headers match
78       *         the supplied qualified name.
79       * @throws WebServiceException If an error occurs when using the supplied
80       *         JAXBContext to unmarshall. The cause of
81       *         the WebServiceException is the original JAXBException.
82       */
83      public Object[] getHeaders(QName header, JAXBContext context,
84          boolean allRoles);
85
86      /** Gets the SOAP actor roles associated with an execution
87       * of the handler chain.
88       * Note that SOAP actor roles apply to the SOAP node and
89       * are managed using {@link javax.xml.ws.soap.SOAPBinding#setRoles} and
```

```

90  * {@link javax.xml.ws.soap.SOAPBinding#getRoles}. <code>Handler</code> instances in
91  * the handler chain use this information about the SOAP actor
92  * roles to process the SOAP header blocks. Note that the
93  * SOAP actor roles are invariant during the processing of
94  * SOAP message through the handler chain.
95  *
96  * @return Array of <code>String</code> for SOAP actor roles
97  **/
98  public Set<String> getRoles();
99 }

```

## 30 Xml Ws Http (javax.xml.ws.http package)

### 30.1 HTTPBinding.java

Secuencia 299: javax/xml/ws/http/HTTPBinding.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.http;
27
28
29 import javax.xml.ws.Binding;
30
31 /** The <code>HTTPBinding</code> interface is an
32  * abstraction for the XML/HTTP binding.
33  *
34  * @since JAX-WS 2.0
35  **/
36 public interface HTTPBinding extends Binding {
37

```



```

38  /**
39   * A constant representing the identity of the XML/HTTP binding.
40   */
41   public static final String HTTP_BINDING = "http://www.w3.org/2004/08/wsdl/http";
42 }

```

## 30.2 HTTPException.java

Secuencia 300: javax/xml/ws/http/HTTPException.java

```

1  /*
2   * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws.http;
27
28
29 /** The <code>HTTPException</code> exception represents a
30  * XML/HTTP fault.
31  *
32  * <p>Since there is no standard format for faults or exceptions
33  * in XML/HTTP messaging, only the HTTP status code is captured.
34  *
35  * @since JAX-WS 2.0
36  */
37 public class HTTPException extends javax.xml.ws.ProtocolException {
38
39     private int statusCode;
40
41     /** Constructor for the HTTPException
42      * @param statusCode <code>int</code> for the HTTP status code
43      */
44     public HTTPException(int statusCode) {

```

```
45     super();
46     this.statusCode = statusCode;
47 }
48
49 /** Gets the HTTP status code.
50  *
51  * @return HTTP status code
52  */
53 public int getStatusCode() {
54     return statusCode;
55 }
56 }
```

## 31 Xml Ws Soap (javax.xml.ws.soap package)

### 31.1 Addressing.java

Secuencia 301: javax/xml/ws/soap/Addressing.java

```
1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.soap;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.ElementType;
31 import java.lang.annotation.Retention;
32 import java.lang.annotation.RetentionPolicy;
33
34 import javax.xml.ws.BindingProvider;
35 import javax.xml.ws.WebServiceRef;
```

```

36 import javax.xml.ws.WebServiceRefs;
37 import javax.xml.ws.WebServiceProvider;
38 import javax.xml.ws.soap.AddressingFeature.Responses;
39 import javax.xml.ws.spi.WebServiceFeatureAnnotation;
40
41 /**
42  * This annotation represents the use of WS-Addressing with either
43  * the SOAP 1.1/HTTP or SOAP 1.2/HTTP binding. Using this annotation
44  * with any other binding is undefined.
45  * <p>
46  * This annotation MUST only be used in conjunction with the
47  * {@link javax.jws.WebService}, {@link WebServiceProvider},
48  * and {@link WebServiceRef} annotations.
49  * When used with a <code>javax.jws.WebService</code> annotation, this
50  * annotation MUST only be used on the service endpoint implementation
51  * class.
52  * When used with a <code>WebServiceRef</code> annotation, this annotation
53  * MUST only be used when a proxy instance is created. The injected SEI
54  * proxy, and endpoint MUST honor the values of the <code>Addressing</code>
55  * annotation.
56  * <p>
57  * This annotation's behaviour is defined by the corresponding feature
58  * {@link AddressingFeature}.
59  *
60  * @since JAX-WS 2.1
61  */
62 @Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
63 @Retention(RetentionPolicy.RUNTIME)
64 @Documented
65 @WebServiceFeatureAnnotation(id=AddressingFeature.ID,bean=AddressingFeature.class)
66 public @interface Addressing {
67     /**
68      * Specifies if this feature is enabled or disabled. If enabled, it means
69      * the endpoint supports WS-Addressing but does not require its use.
70      * Corresponding
71      * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyaddressing">
72      * 3.1.1 Addressing Assertion</a> must be generated in the generated WSDL.
73      */
74     boolean enabled() default true;
75
76     /**
77      * If addressing is enabled, this property determines whether the endpoint
78      * requires WS-Addressing. If required is true, the endpoint requires
79      * WS-Addressing and WS-Addressing headers MUST
80      * be present on incoming messages. A corresponding
81      * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyaddressing">
82      * 3.1.1 Addressing Assertion</a> must be generated in the WSDL.
83      */
84     boolean required() default false;
85
86     /**
87      * If addressing is enabled, this property determines whether endpoint
88      * requires the use of anonymous responses, or non-anonymous responses,

```

```

89      * or all.
90      *
91      * <p>
92      * {@link Responses#ALL} supports all response types and this is the
93      * default value.
94      *
95      * <p>
96      * {@link Responses#ANONYMOUS} requires the use of only anonymous
97      * responses. It will result into wsam:AnonymousResponses nested assertion
98      * as specified in
99      * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyanonresponses">
100     * 3.1.2 AnonymousResponses Assertion</a> in the generated WSDL.
101     *
102     * <p>
103     * {@link Responses#NON_ANONYMOUS} requires the use of only non-anonymous
104     * responses. It will result into
105     * wsam:NonAnonymousResponses nested assertion as specified in
106     * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicynonanonresponses">
107     * 3.1.3 NonAnonymousResponses Assertion</a> in the generated WSDL.
108     *
109     * @since JAX-WS 2.2
110     */
111     Responses responses() default Responses.ALL;
112
113 }

```

## 31.2 AddressingFeature.java

Secuencia 302: javax/xml/ws/soap/AddressingFeature.java

```

1  /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */

```

```
25
26 package javax.xml.ws.soap;
27
28 import javax.xml.ws.WebServiceFeature;
29 import javax.xml.ws.Endpoint;
30 import javax.xml.ws.Service;
31
32 /**
33  * AddressingFeature represents the use of WS-Addressing with either
34  * the SOAP 1.1/HTTP or SOAP 1.2/HTTP binding. Using this feature
35  * with any other binding is undefined.
36  * <p>
37  * This feature can be used during the creation of SEI proxy, and
38  * {@link javax.xml.ws.Dispatch} instances on the client side and {@link Endpoint}
39  * instances on the server side. This feature cannot be used for {@link Service}
40  * instance creation on the client side.
41  * <p>
42  * The following describes the effects of this feature with respect
43  * to be enabled or disabled:
44  * <ul>
45  * <li> ENABLED: In this Mode, WS-Addressing will be enabled. It means
46  *     the endpoint supports WS-Addressing but does not require its use.
47  *     A sender could send messages with WS-Addressing headers or without
48  *     WS-Addressing headers. But a receiver MUST consume both types of
49  *     messages.
50  * <li> DISABLED: In this Mode, WS-Addressing will be disabled.
51  *     At runtime, WS-Addressing headers MUST NOT be used by a sender or
52  *     receiver.
53  * </ul>
54  * <p>
55  * If the feature is enabled, the <code>required</code> property determines
56  * whether the endpoint requires WS-Addressing. If it is set true,
57  * WS-Addressing headers MUST be present on incoming and outgoing messages.
58  * By default the <code>required</code> property is <code>>false</code>.
59  *
60  * <p>
61  * If the web service developer has not explicitly enabled this feature,
62  * WSDL's wsam:Addressing policy assertion is used to find
63  * the use of WS-Addressing. By using the feature explicitly, an application
64  * overrides WSDL's indication of the use of WS-Addressing. In some cases,
65  * this is really required. For example, if an application has implemented
66  * WS-Addressing itself, it can use this feature to disable addressing. That
67  * means a JAX-WS implementation doesn't consume or produce WS-Addressing
68  * headers.
69  *
70  * <p>
71  * If addressing is enabled, a corresponding wsam:Addressing policy assertion
72  * must be generated in the WSDL as per
73  * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyassertions">
74  * 3.1 WS-Policy Assertions</a>
75  *
76  * <p>
77  * <b>Example 1: </b>Possible Policy Assertion in the generated WSDL for
```

```

78 * <code>&#64;Addressing</code>
79 * <pre>
80 *   <wsam:Addressing wsp:Optional="true">
81 *     <wsp:Policy/>
82 *   </wsam:Addressing>
83 * </pre>
84 *
85 * <p>
86 * <b>Example 2: </b>Possible Policy Assertion in the generated WSDL for
87 * <code>&#64;Addressing(required=true)</code>
88 * <pre>
89 *   <wsam:Addressing>
90 *     <wsp:Policy/>
91 *   </wsam:Addressing>
92 * </pre>
93 *
94 * <p>
95 * <b>Example 3: </b>Possible Policy Assertion in the generated WSDL for
96 * <code>&#64;Addressing(required=true, responses=Responses.ANONYMOUS)</code>
97 * <pre>
98 *   <wsam:Addressing>
99 *     <wsp:Policy>
100 *       <wsam:AnonymousResponses/>
101 *     </wsp:Policy>
102 *   </wsam:Addressing>
103 * </pre>
104 *
105 * <p>
106 * See <a href="http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/">
107 * Web Services Addressing - Core</a>,
108 * <a href="http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/">
109 * Web Services Addressing 1.0 - SOAP Binding</a>,
110 * and <a href="http://www.w3.org/TR/ws-addr-metadata/">
111 * Web Services Addressing 1.0 - Metadata</a>
112 * for more information on WS-Addressing.
113 *
114 * @see Addressing
115 * @since JAX-WS 2.1
116 */
117
118 public final class AddressingFeature extends WebServiceFeature {
119     /**
120      * Constant value identifying the AddressingFeature
121      */
122     public static final String ID = "http://www.w3.org/2005/08/addressing/module";
123
124     /**
125      * If addressing is enabled, this property determines whether the endpoint
126      * requires WS-Addressing. If required is true, WS-Addressing headers MUST
127      * be present on incoming and outgoing messages.
128      */
129     // should be private final, keeping original modifier due to backwards compatibility
130     protected boolean required;

```

```

131  /**
132  * If addressing is enabled, this property determines if endpoint requires
133  * the use of only anonymous responses, or only non-anonymous responses, or all.
134  *
135  * <p>
136  * {@link Responses#ALL} supports all response types and this is the default
137  * value.
138  *
139  * <p>
140  * {@link Responses#ANONYMOUS} requires the use of only anonymous
141  * responses. It will result into wsam:AnonymousResponses nested assertion
142  * as specified in
143  * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyanonresponses">
144  * 3.1.2 AnonymousResponses Assertion</a> in the generated WSDL.
145  *
146  * <p>
147  * {@link Responses#NON_ANONYMOUS} requires the use of only non-anonymous
148  * responses. It will result into
149  * wsam:NonAnonymousResponses nested assertion as specified in
150  * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicynonanonresponses">
151  * 3.1.3 NonAnonymousResponses Assertion</a> in the generated WSDL.
152  *
153  * @since JAX-WS 2.2
154  */
155  public enum Responses {
156      /**
157       * Specifies the use of only anonymous
158       * responses. It will result into wsam:AnonymousResponses nested assertion
159       * as specified in
160       * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyanonresponses">
161       * 3.1.2 AnonymousResponses Assertion</a> in the generated WSDL.
162       */
163      ANONYMOUS,
164
165      /**
166       * Specifies the use of only non-anonymous
167       * responses. It will result into
168       * wsam:NonAnonymousResponses nested assertion as specified in
169       * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicynonanonresponses">
170       * 3.1.3 NonAnonymousResponses Assertion</a> in the generated WSDL.
171       */
172      NON_ANONYMOUS,
173
174      /**
175       * Supports all response types and this is the default
176       */
177      ALL
178  }
179
180  private final Responses responses;
181
182  /**

```

```
184     * Creates and configures an <code>AddressingFeature</code> with the
185     * use of addressing requirements. The created feature enables
186     * ws-addressing i.e. supports ws-addressing but doesn't require
187     * its use. It is also configured to accept all the response types.
188     */
189     public AddressingFeature() {
190         this(true, false, Responses.ALL);
191     }
192
193     /**
194     * Creates and configures an <code>AddressingFeature</code> with the
195     * use of addressing requirements. If <code>enabled</code> is true,
196     * it enables ws-addressing i.e. supports ws-addressing but doesn't
197     * require its use. It also configures to accept all the response types.
198     *
199     * @param enabled true enables ws-addressing i.e.ws-addressing
200     * is supported but doesn't require its use
201     */
202     public AddressingFeature(boolean enabled) {
203         this(enabled, false, Responses.ALL);
204     }
205
206     /**
207     * Creates and configures an <code>AddressingFeature</code> with the
208     * use of addressing requirements. If <code>enabled</code> and
209     * <code>required</code> are true, it enables ws-addressing and
210     * requires its use. It also configures to accept all the response types.
211     *
212     * @param enabled true enables ws-addressing i.e.ws-addressing
213     * is supported but doesn't require its use
214     * @param required true means requires the use of ws-addressing .
215     */
216     public AddressingFeature(boolean enabled, boolean required) {
217         this(enabled, required, Responses.ALL);
218     }
219
220     /**
221     * Creates and configures an <code>AddressingFeature</code> with the
222     * use of addressing requirements. If <code>enabled</code> and
223     * <code>required</code> are true, it enables ws-addressing and
224     * requires its use. Also, the response types can be configured using
225     * <code>responses</code> parameter.
226     *
227     * @param enabled true enables ws-addressing i.e.ws-addressing
228     * is supported but doesn't require its use
229     * @param required true means requires the use of ws-addressing .
230     * @param responses specifies what type of responses are required
231     *
232     * @since JAX-WS 2.2
233     */
234     public AddressingFeature(boolean enabled, boolean required, Responses responses) {
235         this.enabled = enabled;
236         this.required = required;
```



```

237         this.responses = responses;
238     }
239
240     /**
241      * {@inheritDoc}
242      */
243     public String getID() {
244         return ID;
245     }
246
247     /**
248      * If addressing is enabled, this property determines whether the endpoint
249      * requires WS-Addressing. If required is true, WS-Addressing headers MUST
250      * be present on incoming and outgoing messages.
251      *
252      * @return the current required value
253      */
254     public boolean isRequired() {
255         return required;
256     }
257
258     /**
259      * If addressing is enabled, this property determines whether endpoint
260      * requires the use of anonymous responses, or non-anonymous responses,
261      * or all responses.
262      *
263      * <p>
264      * @return {@link Responses#ALL} when endpoint supports all types of
265      * responses,
266      *         {@link Responses#ANONYMOUS} when endpoint requires the use of
267      * only anonymous responses,
268      *         {@link Responses#NON_ANONYMOUS} when endpoint requires the use
269      * of only non-anonymous responses
270      *
271      * @since JAX-WS 2.2
272      */
273     public Responses getResponses() {
274         return responses;
275     }
276
277 }

```

### 31.3 MTOM.java

Secuencia 303: javax/xml/ws/soap/MTOM.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.soap;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.ElementType;
31 import java.lang.annotation.Retention;
32 import java.lang.annotation.RetentionPolicy;
33
34 import javax.xml.ws.spi.WebServiceFeatureAnnotation;
35 import javax.xml.ws.WebServiceRef;
36 import javax.xml.ws.WebServiceProvider;
37
38 /**
39  * This feature represents the use of MTOM with a
40  * web service.
41  * <p>
42  * This annotation MUST only be used in conjunction the
43  * <code>javax.jws.WebService</code>, {@link WebServiceProvider},
44  * {@link WebServiceRef} annotations.
45  * When used with the <code>javax.jws.WebService</code> annotation this
46  * annotation MUST only be used on the service endpoint implementation
47  * class.
48  * When used with a <code>WebServiceRef</code> annotation, this annotation
49  * MUST only be used when a proxy instance is created. The injected SEI
50  * proxy, and endpoint MUST honor the values of the <code>MTOM</code>
51  * annotation.
52  * <p>
53  *
54  * This annotation's behaviour is defined by the corresponding feature
55  * {@link MTOMFeature}.
56  *
57  * @since JAX-WS 2.1
58  */
59 @Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
60 @Retention(RetentionPolicy.RUNTIME)
61 @Documented
```

```
62 @WebServiceFeatureAnnotation(id=MTOMFeature.ID,bean=MTOMFeature.class)
63 public @interface MTOM {
64     /**
65      * Specifies if this feature is enabled or disabled.
66      */
67     boolean enabled() default true;
68
69     /**
70      * Property for MTOM threshold value. When MTOM is enabled, binary data above this
71      * size in bytes will be XOP encoded or sent as attachment. The value of this property
72      * MUST always be >= 0. Default value is 0.
73      */
74     int threshold() default 0;
75 }
```

### 31.4 MTOMFeature.java

Secuencia 304: javax/xml/ws/soap/MTOMFeature.java

```
1  /*
2   * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws.soap;
27
28 import javax.xml.ws.WebServiceFeature;
29 import javax.xml.ws.WebServiceException;
30 import javax.xml.ws.Endpoint;
31 import javax.xml.ws.Service;
32
33 /**
34  * This feature represents the use of MTOM with a
35  * web service.
```

```

36  *
37  * This feature can be used during the creation of SEI proxy, and
38  * {@link javax.xml.ws.Dispatch} instances on the client side and {@link Endpoint}
39  * instances on the server side. This feature cannot be used for {@link Service}
40  * instance creation on the client side.
41  *
42  * <p>
43  * The following describes the affects of this feature with respect
44  * to being enabled or disabled:
45  * <ul>
46  * <li> ENABLED: In this Mode, MTOM will be enabled. A receiver MUST accept
47  * both a non-optimized and an optimized message, and a sender MAY send an
48  * optimized message, or a non-optimized message. The heuristics used by a
49  * sender to determine whether to use optimization or not are
50  * implementation-specific.
51  * <li> DISABLED: In this Mode, MTOM will be disabled
52  * </ul>
53  * <p>
54  * The {@link #threshold} property can be used to set the threshold
55  * value used to determine when binary data should be XOP encoded.
56  *
57  * @since JAX-WS 2.1
58  */
59  public final class MTOMFeature extends WebServiceFeature {
60      /**
61       * Constant value identifying the MTOMFeature
62       */
63      public static final String ID = "http://www.w3.org/2004/08/soap/features/http-optimization";
64
65
66      /**
67       * Property for MTOM threshold value. This property serves as a hint when
68       * MTOM is enabled, binary data above this size in bytes SHOULD be sent
69       * as attachment.
70       * The value of this property MUST always be >= 0. Default value is 0.
71       */
72      // should be changed to private final, keeping original modifier to keep backwards
       compatibility
73      protected int threshold;
74
75
76      /**
77       * Create an <code>MTOMFeature</code>.
78       * The instance created will be enabled.
79       */
80      public MTOMFeature() {
81          this.enabled = true;
82          this.threshold = 0;
83      }
84
85      /**
86       * Creates an <code>MTOMFeature</code>.
87       *

```

```
88     * @param enabled specifies if this feature should be enabled or not
89     */
90     public MTOMFeature(boolean enabled) {
91         this.enabled = enabled;
92         this.threshold = 0;
93     }
94
95
96     /**
97     * Creates an <code>MTOMFeature</code>.
98     * The instance created will be enabled.
99     *
100    * @param threshold the size in bytes that binary data SHOULD be before
101    * being sent as an attachment.
102    *
103    * @throws WebServiceException if threshold is < 0
104    */
105    public MTOMFeature(int threshold) {
106        if (threshold < 0)
107            throw new WebServiceException("MTOMFeature.threshold must be >= 0, actual value: "+
108                threshold);
109        this.enabled = true;
110        this.threshold = threshold;
111    }
112
113    /**
114    * Creates an <code>MTOMFeature</code>.
115    *
116    * @param enabled specifies if this feature should be enabled or not
117    * @param threshold the size in bytes that binary data SHOULD be before
118    * being sent as an attachment.
119    *
120    * @throws WebServiceException if threshold is < 0
121    */
122    public MTOMFeature(boolean enabled, int threshold) {
123        if (threshold < 0)
124            throw new WebServiceException("MTOMFeature.threshold must be >= 0, actual value: "+
125                threshold);
126        this.enabled = enabled;
127        this.threshold = threshold;
128    }
129
130    /**
131    * {@inheritDoc}
132    */
133    public String getID() {
134        return ID;
135    }
136
137    /**
138    * Gets the threshold value used to determine when binary data
139    * should be sent as an attachment.
140    *
```

```
139     * @return the current threshold size in bytes
140     */
141     public int getThreshold() {
142         return threshold;
143     }
144 }
```

### 31.5 SOAPBinding.java

Secuencia 305: javax/xml/ws/soap/SOAPBinding.java

```
1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.soap;
27
28
29 import java.util.Set;
30 import javax.xml.ws.Binding;
31 import javax.xml.soap.SOAPFactory;
32 import javax.xml.soap.MessageFactory;
33
34 /** The <code>SOAPBinding</code> interface is an abstraction for
35  * the SOAP binding.
36  *
37  * @since JAX-WS 2.0
38  */
39 public interface SOAPBinding extends Binding {
40
41     /**
42      * A constant representing the identity of the SOAP 1.1 over HTTP binding.
43      */
44 }
```

```
44 public static final String SOAP11HTTP_BINDING = "http://schemas.xmlsoap.org/wsdl/soap/http";
45
46 /**
47  * A constant representing the identity of the SOAP 1.2 over HTTP binding.
48  */
49 public static final String SOAP12HTTP_BINDING = "http://www.w3.org/2003/05/soap/bindings/HTTP/";
50
51 /**
52  * A constant representing the identity of the SOAP 1.1 over HTTP binding
53  * with MTOM enabled by default.
54  */
55 public static final String SOAP11HTTP_MTOM_BINDING = "http://schemas.xmlsoap.org/wsdl/soap/http?
    mtom=true";
56
57 /**
58  * A constant representing the identity of the SOAP 1.2 over HTTP binding
59  * with MTOM enabled by default.
60  */
61 public static final String SOAP12HTTP_MTOM_BINDING = "http://www.w3.org/2003/05/soap/bindings/
    HTTP/?mtom=true";
62
63
64 /** Gets the roles played by the SOAP binding instance.
65  *
66  * @return Set<String> The set of roles played by the binding instance.
67  */
68 public Set<String> getRoles();
69
70 /** Sets the roles played by the SOAP binding instance.
71  *
72  * @param roles The set of roles played by the binding instance.
73  * @throws WebServiceException On an error in the configuration of
74  * the list of roles.
75  */
76 public void setRoles(Set<String> roles);
77
78 /**
79  * Returns <code>true</code> if the use of MTOM is enabled.
80  *
81  * @return <code>true</code> if and only if the use of MTOM is enabled.
82  */
83
84 public boolean isMTOMEnabled();
85
86 /**
87  * Enables or disables use of MTOM.
88  *
89  * @param flag A <code>boolean</code> specifying whether the use of MTOM should
90  * be enabled or disabled.
91  * @throws WebServiceException If the specified setting is not supported
92  * by this binding instance.
93  *
94  */
```

```

95 public void setMTOMEnabled(boolean flag);
96
97 /**
98  * Gets the SAAJ <code>SOAPFactory</code> instance used by this SOAP binding.
99  *
100  * @return SOAPFactory instance used by this SOAP binding.
101  */
102 public SOAPFactory getSOAPFactory();
103
104 /**
105  * Gets the SAAJ <code>MessageFactory</code> instance used by this SOAP binding.
106  *
107  * @return MessageFactory instance used by this SOAP binding.
108  */
109 public MessageFactory getMessageFactory();
110 }

```

### 31.6 SOAPFaultException.java

Secuencia 306: javax/xml/ws/soap/SoapFaultException.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.soap;
27
28 import javax.xml.ws.soap.SOAPFault;
29
30 /** The <code>SOAPFaultException</code> exception represents a
31  * SOAP 1.1 or 1.2 fault.
32  *
33  * <p>A <code>SOAPFaultException</code> wraps a SAAJ <code>SOAPFault</code>

```



```

34 * that manages the SOAP-specific representation of faults.
35 * The <code>createFault</code> method of
36 * <code>javax.xml.soap.SOAPFactory</code> may be used to create an instance
37 * of <code>javax.xml.soap.SOAPFault</code> for use with the
38 * constructor. <code>SOAPBinding</code> contains an accessor for the
39 * <code>SOAPFactory</code> used by the binding instance.
40 *
41 * <p>Note that the value of <code>getFault</code> is the only part of the
42 * exception used when searializing a SOAP fault.
43 *
44 * <p>Refer to the SOAP specification for a complete
45 * description of SOAP faults.
46 *
47 * @see javax.xml.soap.SOAPFault
48 * @see javax.xml.ws.soap.SOAPBinding#getSOAPFactory
49 * @see javax.xml.ws.ProtocolException
50 *
51 * @since JAX-WS 2.0
52 **/
53 public class SOAPFaultException extends javax.xml.ws.ProtocolException {
54
55     private SOAPFault fault;
56
57     /** Constructor for SOAPFaultException
58      * @param fault <code>SOAPFault</code> representing the fault
59      *
60      * @see javax.xml.soap.SOAPFactory#createFault
61      **/
62     public SOAPFaultException(SOAPFault fault) {
63         super(fault.getFaultString());
64         this.fault = fault;
65     }
66
67     /** Gets the embedded <code>SOAPFault</code> instance.
68      *
69      * @return <code>javax.xml.soap.SOAPFault</code> SOAP
70      *         fault element
71      **/
72     public javax.xml.soap.SOAPFault getFault() {
73         return this.fault;
74     }
75 }

```

## 32 Xml Ws Spi (javax.xml.ws.spi package)

### 32.1 FactoryFinder.java

Secuencia 307: javax/xml/ws/spi/FactoryFinder.java

```

1 /*
2  * Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *

```

```
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.spi;
27
28 import java.io.*;
29
30 import java.util.Properties;
31 import javax.xml.ws.WebServiceException;
32
33 class FactoryFinder {
34
35     /**
36      * Creates an instance of the specified class using the specified
37      * <code>ClassLoader</code> object.
38      *
39      * @exception WebServiceException if the given class could not be found
40      *         or could not be instantiated
41      */
42     private static Object newInstance(String className,
43                                     ClassLoader classLoader)
44     {
45         try {
46             Class spiClass = safeLoadClass(className, classLoader);
47             return spiClass.newInstance();
48         } catch (ClassNotFoundException x) {
49             throw new WebServiceException(
50                 "Provider " + className + " not found", x);
51         } catch (Exception x) {
52             throw new WebServiceException(
53                 "Provider " + className + " could not be instantiated: " + x,
54                 x);
55         }
56     }
57
58     /**
```

```

59  * Finds the implementation <code>Class</code> object for the given
60  * factory name, or if that fails, finds the <code>Class</code> object
61  * for the given fallback class name. The arguments supplied MUST be
62  * used in order. If using the first argument is successful, the second
63  * one will not be used.
64  * <P>
65  * This method is package private so that this code can be shared.
66  *
67  * @return the <code>Class</code> object of the specified message factory;
68  *         may not be <code>null</code>
69  *
70  * @param factoryId      the name of the factory to find, which is
71  *                       a system property
72  * @param fallbackClassName the implementation class name, which is
73  *                       to be used only if nothing else
74  *                       is found; <code>null</code> to indicate that
75  *                       there is no fallback class name
76  * @exception WebServiceException if there is an error
77  */
78  static Object find(String factoryId, String fallbackClassName)
79  {
80      if (isOsgi()) {
81          return lookupUsingOSGiServiceLoader(factoryId);
82      }
83      ClassLoader classLoader;
84      try {
85          classLoader = Thread.currentThread().getContextClassLoader();
86      } catch (Exception x) {
87          throw new WebServiceException(x.toString(), x);
88      }
89
90      String serviceId = "META-INF/services/" + factoryId;
91      // try to find services in CLASSPATH
92      BufferedReader rd = null;
93      try {
94          InputStream is;
95          if (classLoader == null) {
96              is=ClassLoader.getResourceAsStream(serviceId);
97          } else {
98              is=classLoader.getResourceAsStream(serviceId);
99          }
100
101          if( is!=null ) {
102              rd = new BufferedReader(new InputStreamReader(is, "UTF-8"));
103
104              String factoryClassName = rd.readLine();
105
106              if (factoryClassName != null &&
107                  ! "".equals(factoryClassName)) {
108                  return newInstance(factoryClassName, classLoader);
109              }
110          }
111      } catch( Exception ignored) {

```

```
112     } finally {
113         close(rd);
114     }
115
116
117     // try to read from $java.home/lib/jaxws.properties
118     FileInputStream inStream = null;
119     try {
120         String javah=System.getProperty( "java.home" );
121         String configFile = javah + File.separator +
122             "lib" + File.separator + "jaxws.properties";
123         File f=new File( configFile );
124         if( f.exists() ) {
125             Properties props=new Properties();
126             inStream = new FileInputStream(f);
127             props.load(inStream);
128             String factoryClassName = props.getProperty(factoryId);
129             return newInstance(factoryClassName, classLoader);
130         }
131     } catch(Exception ignored) {
132     } finally {
133         close(inStream);
134     }
135
136     // Use the system property
137     try {
138         String systemProp =
139             System.getProperty( factoryId );
140         if( systemProp!=null ) {
141             return newInstance(systemProp, classLoader);
142         }
143     } catch (SecurityException ignored) {
144     }
145
146     if (fallbackClassName == null) {
147         throw new WebServiceException(
148             "Provider for " + factoryId + " cannot be found", null);
149     }
150
151     return newInstance(fallbackClassName, classLoader);
152 }
153
154 private static void close(Closeable closeable) {
155     if (closeable != null) {
156         try {
157             closeable.close();
158         } catch (IOException ignored) {
159         }
160     }
161 }
162
163
164 /**
```

```
165     * Loads the class, provided that the calling thread has an access to the class being loaded.
166     */
167     private static Class safeLoadClass(String className, ClassLoader classLoader) throws
        ClassNotFoundException {
168         try {
169             // make sure that the current thread has an access to the package of the given name.
170             SecurityManager s = System.getSecurityManager();
171             if (s != null) {
172                 int i = className.lastIndexOf('.');
173                 if (i != -1) {
174                     s.checkPackageAccess(className.substring(0, i));
175                 }
176             }
177
178             if (classLoader == null)
179                 return Class.forName(className);
180             else
181                 return classLoader.loadClass(className);
182         } catch (SecurityException se) {
183             // anyone can access the platform default factory class without permission
184             if (Provider.DEFAULT_JAXWSPROVIDER.equals(className))
185                 return Class.forName(className);
186             throw se;
187         }
188     }
189
190     private static final String OSGI_SERVICE_LOADER_CLASS_NAME = "com.sun.org.glassfish.hk2.
        osgiresourcelocator.ServiceLoader";
191
192     private static boolean isOsgi() {
193         try {
194             Class.forName(OSGI_SERVICE_LOADER_CLASS_NAME);
195             return true;
196         } catch (ClassNotFoundException ignored) {
197         }
198         return false;
199     }
200
201     private static Object lookupUsingOSGiServiceLoader(String factoryId) {
202         try {
203             // Use reflection to avoid having any dependency on ServiceLoader class
204             Class serviceClass = Class.forName(factoryId);
205             Class[] args = new Class[] { serviceClass };
206             Class target = Class.forName(OSGI_SERVICE_LOADER_CLASS_NAME);
207             java.lang.reflect.Method m = target.getMethod("lookupProviderInstances", Class.class);
208             java.util.Iterator iter = ((Iterable) m.invoke(null, (Object[]) args)).iterator();
209             return iter.hasNext() ? iter.next() : null;
210         } catch (Exception ignored) {
211             // log and continue
212             return null;
213         }
214     }
215 }
```

216 }

## 32.2 Invoker.java

Secuencia 308: javax/xml/ws/spi/Invoker.java

```
1  /*
2  * Copyright (c) 2009, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.spi;
27
28 import javax.xml.ws.WebServiceContext;
29 import javax.xml.ws.WebServiceFeature;
30 import java.lang.reflect.Method;
31 import java.lang.reflect.InvocationTargetException;
32
33 /**
34 * Invoker hides the detail of calling into application endpoint
35 * implementation. Container hands over an implementation of Invoker
36 * to JAX-WS runtime, and jax-ws runtime calls {@link #invoke}
37 * for a web service invocation. Finally, Invoker does the actual
38 * invocation of web service on endpoint instance.
39 *
40 * Container also injects the provided <code>WebServiceContext</code> and takes
41 * care of invoking <code>javax.annotation.PostConstruct</code> methods,
42 * if present, on the endpoint implementation.
43 *
44 * @see Provider#createEndpoint(String, Class, Invoker, WebServiceFeature...)
45 * @author Jitendra Kotamraju
46 * @since JAX-WS 2.2
47 */
48
```

```

49 public abstract class Invoker {
50
51     /**
52      * JAX-WS runtimes calls this method to ask container to inject
53      * WebServiceContext on the endpoint instance. The
54      * <code>WebServiceContext</code> object uses thread-local information
55      * to return the correct information during the actual endpoint invocation
56      * regardless of how many threads are concurrently being used to serve
57      * requests.
58      *
59      * @param webServiceContext a holder for MessageContext
60      * @throws IllegalAccessException if the injection done
61      *         by reflection API throws this exception
62      * @throws IllegalArgumentException if the injection done
63      *         by reflection API throws this exception
64      * @throws InvocationTargetException if the injection done
65      *         by reflection API throws this exception
66      */
67     public abstract void inject(WebServiceContext webServiceContext)
68     throws IllegalAccessException, IllegalArgumentException, InvocationTargetException;
69
70     /**
71      * JAX-WS runtime calls this method to do the actual web service
72      * invocation on endpoint instance. The injected
73      * <code>WebServiceContext.getMessageContext()</code> gives the correct
74      * information for this invocation.
75      *
76      * @param m Method to be invoked on the service
77      * @param args Method arguments
78      * @return return value of the method
79      * @throws IllegalAccessException if the invocation done
80      *         by reflection API throws this exception
81      * @throws IllegalArgumentException if the invocation done
82      *         by reflection API throws this exception
83      * @throws InvocationTargetException if the invocation done
84      *         by reflection API throws this exception
85
86      * @see Method#invoke
87      */
88     public abstract Object invoke(Method m, Object... args)
89     throws IllegalAccessException, IllegalArgumentException, InvocationTargetException;
90
91 }

```

### 32.3 Provider.java

Secuencia 309: javax/xml/ws/spi/Provider.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.spi;
27
28 import java.net.URL;
29 import java.util.List;
30 import java.util.Iterator;
31 import java.util.Map;
32 import java.lang.reflect.Method;
33 import javax.xml.namespace.QName;
34 import javax.xml.ws.*;
35 import javax.xml.ws.wsaddressing.W3CEndpointReference;
36
37 import org.w3c.dom.Element;
38
39 /**
40  * Service provider for <code>ServiceDelegate</code> and
41  * <code>Endpoint</code> objects.
42  * <p>
43  *
44  * @since JAX-WS 2.0
45  */
46 public abstract class Provider {
47
48     /**
49      * A constant representing the property used to lookup the
50      * name of a <code>Provider</code> implementation
51      * class.
52      */
53     static public final String JAXWSPROVIDER_PROPERTY
54         = "javax.xml.ws.spi.Provider";
55
56     /**
57      * A constant representing the name of the default
58      * <code>Provider</code> implementation class.
59      */
60 }
```



```

60 // Using two strings so that package renaming doesn't change it
61 static final String DEFAULT_JAXWS_PROVIDER
62     = "com.sun"+"xml.internal.ws.spi.ProviderImpl";
63
64 /**
65  * Take advantage of Java SE 6's java.util.ServiceLoader API.
66  * Using reflection so that there is no compile-time dependency on SE 6.
67  */
68 static private final Method loadMethod;
69 static private final Method iteratorMethod;
70 static {
71     Method tLoadMethod = null;
72     Method tIteratorMethod = null;
73     try {
74         Class<?> clazz = Class.forName("java.util.ServiceLoader");
75         tLoadMethod = clazz.getMethod("load", Class.class);
76         tIteratorMethod = clazz.getMethod("iterator");
77     } catch (ClassNotFoundException ce) {
78         // Running on Java SE 5
79     } catch (NoSuchMethodException ne) {
80         // Shouldn't happen
81     }
82     loadMethod = tLoadMethod;
83     iteratorMethod = tIteratorMethod;
84 }
85
86
87 /**
88  * Creates a new instance of Provider
89  */
90 protected Provider() {
91 }
92
93 /**
94  *
95  * Creates a new provider object.
96  * <p>
97  * The algorithm used to locate the provider subclass to use consists
98  * of the following steps:
99  * <p>
100  * <ul>
101  * <li>
102  * If a resource with the name of
103  * <code>META-INF/services/javax.xml.ws.spi.Provider</code>
104  * exists, then its first line, if present, is used as the UTF-8 encoded
105  * name of the implementation class.
106  * </li>
107  * <li>
108  * If the $java.home/lib/jaxws.properties file exists and it is readable by
109  * the <code>java.util.Properties.load(InputStream)</code> method and it contains
110  * an entry whose key is <code>javax.xml.ws.spi.Provider</code>, then the value of
111  * that entry is used as the name of the implementation class.
112  * </li>

```

```

113     * <li>
114     *   If a system property with the name <code>javax.xml.ws.spi.Provider</code>
115     *   is defined, then its value is used as the name of the implementation class.
116     * </li>
117     * <li>
118     *   Finally, a default implementation class name is used.
119     * </li>
120     * </ul>
121     *
122     */
123     public static Provider provider() {
124         try {
125             Object provider = getProviderUsingServiceLoader();
126             if (provider == null) {
127                 provider = FactoryFinder.find(JAXWS PROVIDER_PROPERTY, DEFAULT_JAXWS PROVIDER);
128             }
129             if (!(provider instanceof Provider)) {
130                 Class pClass = Provider.class;
131                 String classnameAsResource = pClass.getName().replace('.', '/') + ".class";
132                 ClassLoader loader = pClass.getClassLoader();
133                 if (loader == null) {
134                     loader = ClassLoader.getSystemClassLoader();
135                 }
136                 URL targetTypeURL = loader.getResource(classnameAsResource);
137                 throw new LinkageError("ClassCastException: attempting to cast" +
138                     provider.getClass().getClassLoader().getResource(classnameAsResource) +
139                     "to" + targetTypeURL.toString() );
140             }
141             return (Provider) provider;
142         } catch (WebServiceException ex) {
143             throw ex;
144         } catch (Exception ex) {
145             throw new WebServiceException("Unable to createEndpointReference Provider", ex);
146         }
147     }
148
149
150     private static Provider getProviderUsingServiceLoader() {
151         if (loadMethod != null) {
152             Object loader;
153             try {
154                 loader = loadMethod.invoke(null, Provider.class);
155             } catch (Exception e) {
156                 throw new WebServiceException("Cannot invoke java.util.ServiceLoader#load()", e);
157             }
158
159             Iterator<Provider> it;
160             try {
161                 it = (Iterator<Provider>) iteratorMethod.invoke(loader);
162             } catch (Exception e) {
163                 throw new WebServiceException("Cannot invoke java.util.ServiceLoader#iterator()", e);
164             }

```

```

165         return it.hasNext() ? it.next() : null;
166     }
167     return null;
168 }
169
170 /**
171  * Creates a service delegate object.
172  * <p>
173  * @param wsdlDocumentLocation A URL pointing to the WSDL document
174  *     for the service, or <code>null</code> if there isn't one.
175  * @param serviceName The qualified name of the service.
176  * @param serviceClass The service class, which MUST be either
177  *     <code>javax.xml.ws.Service</code> or a subclass thereof.
178  * @return The newly created service delegate.
179  */
180 public abstract ServiceDelegate createServiceDelegate(
181     java.net.URL wsdlDocumentLocation,
182     QName serviceName, Class<? extends Service> serviceClass);
183
184 /**
185  * Creates a service delegate object.
186  * <p>
187  * @param wsdlDocumentLocation A URL pointing to the WSDL document
188  *     for the service, or <code>null</code> if there isn't one.
189  * @param serviceName The qualified name of the service.
190  * @param serviceClass The service class, which MUST be either
191  *     <code>javax.xml.ws.Service</code> or a subclass thereof.
192  * @param features Web Service features that must be configured on
193  *     the service. If the provider doesn't understand a feature,
194  *     it must throw a WebServiceException.
195  * @return The newly created service delegate.
196  *
197  * @since JAX-WS 2.2
198  */
199 public ServiceDelegate createServiceDelegate(
200     java.net.URL wsdlDocumentLocation,
201     QName serviceName, Class<? extends Service> serviceClass, WebServiceFeature ...
202     features) {
203     throw new UnsupportedOperationException("JAX-WS 2.2 implementation must override this
204         default behaviour.");
205 }
206
207 /**
208  *
209  * Creates an endpoint object with the provided binding and implementation
210  * object.
211  *
212  * @param bindingId A URI specifying the desired binding (e.g. SOAP/HTTP)
213  * @param implementor A service implementation object to which
214  *     incoming requests will be dispatched. The corresponding
215  *     class MUST be annotated with all the necessary Web service
216  *     annotations.

```

```

216     * @return The newly created endpoint.
217     */
218     public abstract Endpoint createEndpoint(String bindingId,
219         Object implementor);
220
221
222     /**
223     * Creates and publishes an endpoint object with the specified
224     * address and implementation object.
225     *
226     * @param address A URI specifying the address and transport/protocol
227     *     to use. A http: URI MUST result in the SOAP 1.1/HTTP
228     *     binding being used. Implementations may support other
229     *     URI schemes.
230     * @param implementor A service implementation object to which
231     *     incoming requests will be dispatched. The corresponding
232     *     class MUST be annotated with all the necessary Web service
233     *     annotations.
234     * @return The newly created endpoint.
235     */
236     public abstract Endpoint createAndPublishEndpoint(String address,
237         Object implementor);
238
239     /**
240     * read an EndpointReference from the infoSet contained in
241     * <code>eprInfoSet</code>.
242     *
243     * @param eprInfoSet infoSet for EndpointReference
244     *
245     * @return the <code>EndpointReference</code> unmarshalled from
246     * <code>eprInfoSet</code>. This method never returns <code>null</code>.
247     *
248     * @throws WebServiceException If there is an error creating the
249     * <code>EndpointReference</code> from the specified <code>eprInfoSet</code>.
250     *
251     * @throws NullPointerException If the <code>null</code>
252     * <code>eprInfoSet</code> value is given.
253     *
254     * @since JAX-WS 2.1
255     */
256     public abstract EndpointReference readEndpointReference(javax.xml.transform.Source eprInfoSet);
257
258
259     /**
260     * The getPort method returns a proxy. If there
261     * are any reference parameters in the
262     * <code>endpointReference</code>, then those reference
263     * parameters MUST appear as SOAP headers, indicating them to be
264     * reference parameters, on all messages sent to the endpoint.
265     * The parameter <code>serviceEndpointInterface</code> specifies
266     * the service endpoint interface that is supported by the
267     * returned proxy.
268     * The parameter <code>endpointReference</code> specifies the

```

```

269 * endpoint that will be invoked by the returned proxy.
270 * In the implementation of this method, the JAX-WS
271 * runtime system takes the responsibility of selecting a protocol
272 * binding (and a port) and configuring the proxy accordingly from
273 * the WSDL metadata of the
274 * <code>serviceEndpointInterface</code> and the <code>EndpointReference</code>.
275 * For this method
276 * to successfully return a proxy, WSDL metadata MUST be available and the
277 * <code>endpointReference</code> MUST contain an implementation understood
278 * <code>serviceName</code> metadata.
279 *
280 *
281 * @param endpointReference the EndpointReference that will
282 * be invoked by the returned proxy.
283 * @param serviceEndpointInterface Service endpoint interface
284 * @param features A list of WebServiceFeatures to configure on the
285 * proxy. Supported features not in the <code>features
286 * </code> parameter will have their default values.
287 * @return Object Proxy instance that supports the
288 * specified service endpoint interface
289 * @throws WebServiceException
290 *
291 * <UL>
292 * <LI>If there is an error during creation
293 * of the proxy
294 * <LI>If there is any missing WSDL metadata
295 * as required by this method}
296 * <LI>If this
297 * <code>endpointReference</code>
298 * is illegal
299 * <LI>If an illegal
300 * <code>serviceEndpointInterface</code>
301 * is specified
302 * <LI>If a feature is enabled that is not compatible with
303 * this port or is unsupported.
304 * </UL>
305 *
306 * @see WebServiceFeature
307 *
308 * @since JAX-WS 2.1
309 */
310 public abstract <T> T getPort(EndpointReference endpointReference,
311 Class<T> serviceEndpointInterface,
312 WebServiceFeature... features);
313
314 /**
315 * Factory method to create a <code>W3CEndpointReference</code>.
316 *
317 * <p>
318 * This method can be used to create a <code>W3CEndpointReference</code>
319 * for any endpoint by specifying the <code>address</code> property along
320 * with any other desired properties. This method
321 * can also be used to create a <code>W3CEndpointReference</code> for
322 * an endpoint that is published by the same Java EE application.

```

```

322 * To do so the <code>address</code> property can be provided or this
323 * method can automatically determine the <code>address</code> of
324 * an endpoint that is published by the same Java EE application and is
325 * identified by the <code>serviceName</code> and
326 * <code>portName</code> properties. If the <code>address</code> is
327 * <code>null</code> and the <code>serviceName</code> and
328 * <code>portName</code> do not identify an endpoint published by the
329 * same Java EE application, a
330 * <code>javax.lang.IllegalStateException</code> MUST be thrown.
331 *
332 * @param address Specifies the address of the target endpoint
333 * @param serviceName Qualified name of the service in the WSDL.
334 * @param portName Qualified name of the endpoint in the WSDL.
335 * @param metadata A list of elements that should be added to the
336 * <code>W3CEndpointReference</code> instances <code>wsa:metadata</code>
337 * element.
338 * @param wsdlDocumentLocation URL for the WSDL document location for
339 * the service.
340 * @param referenceParameters Reference parameters to be associated
341 * with the returned <code>EndpointReference</code> instance.
342 *
343 * @return the <code>W3CEndpointReference</code> created from
344 *         <code>serviceName</code>, <code>portName</code>,
345 *         <code>metadata</code>, <code>wsdlDocumentLocation</code>
346 *         and <code>referenceParameters</code>. This method
347 *         never returns <code>null</code>.
348 *
349 * @throws java.lang.IllegalStateException
350 *         <ul>
351 *         <li>If the <code>address</code>, <code>serviceName</code> and
352 *         <code>portName</code> are all <code>null</code>.
353 *         <li>If the <code>serviceName</code> service is <code>null</code> and the
354 *         <code>portName</code> is NOT <code>null</code>.
355 *         <li>If the <code>address</code> property is <code>null</code> and
356 *         the <code>serviceName</code> and <code>portName</code> do not
357 *         specify a valid endpoint published by the same Java EE
358 *         application.
359 *         <li>If the <code>serviceName</code> is NOT <code>null</code>
360 *         and is not present in the specified WSDL.
361 *         <li>If the <code>portName</code> port is not <code>null</code> and it
362 *         is not present in <code>serviceName</code> service in the WSDL.
363 *         <li>If the <code>wsdlDocumentLocation</code> is NOT <code>null</code>
364 *         and does not represent a valid WSDL.
365 *         </ul>
366 * @throws WebServiceException If an error occurs while creating the
367 *         <code>W3CEndpointReference</code>.
368 *
369 * @since JAX-WS 2.1
370 */
371 public abstract W3CEndpointReference createW3CEndpointReference(String address, QName
    serviceName, QName portName,
372     List<Element> metadata, String wsdlDocumentLocation, List<Element> referenceParameters)
    ;

```

```

373
374
375 /**
376  * Factory method to create a W3CEndpointReference.
377  * Using this method, a W3CEndpointReference instance
378  * can be created with extension elements, and attributes.
379  * Provider implementations must override the default
380  * implementation.
381  *
382  * <p>
383  * This method can be used to create a W3CEndpointReference
384  * for any endpoint by specifying the address property along
385  * with any other desired properties. This method
386  * can also be used to create a W3CEndpointReference for
387  * an endpoint that is published by the same Java EE application.
388  * To do so the address property can be provided or this
389  * method can automatically determine the address of
390  * an endpoint that is published by the same Java EE application and is
391  * identified by the serviceName and
392  * portName properties. If the address is
393  * null and the serviceName and
394  * portName do not identify an endpoint published by the
395  * same Java EE application, a
396  * javax.lang.IllegalStateException MUST be thrown.
397  *
398  * @param address Specifies the address of the target endpoint
399  * @param interfaceName the wsam:InterfaceName element in the
400  * wsa:Metadata element.
401  * @param serviceName Qualified name of the service in the WSDL.
402  * @param portName Qualified name of the endpoint in the WSDL.
403  * @param metadata A list of elements that should be added to the
404  * W3CEndpointReference instances wsa:metadata
405  * element.
406  * @param wsdlDocumentLocation URL for the WSDL document location for
407  * the service.
408  * @param referenceParameters Reference parameters to be associated
409  * with the returned EndpointReference instance.
410  * @param elements extension elements to be associated
411  * with the returned EndpointReference instance.
412  * @param attributes extension attributes to be associated
413  * with the returned EndpointReference instance.
414  *
415  * @return the W3CEndpointReference created from
416  *         serviceName, portName,
417  *         metadata, wsdlDocumentLocation
418  *         and referenceParameters. This method
419  *         never returns null.
420  *
421  * @throws java.lang.IllegalStateException
422  *         <ul>
423  *         <li>If the address, serviceName and
424  *         portName are all null.
425  *         <li>If the serviceName service is null and the

```

```

426 *      <code>portName</code> is NOT <code>null</code>.
427 *      <li>If the <code>address</code> property is <code>null</code> and
428 *      the <code>serviceName</code> and <code>portName</code> do not
429 *      specify a valid endpoint published by the same Java EE
430 *      application.
431 *      <li>If the <code>serviceName</code> is NOT <code>null</code>
432 *      and is not present in the specified WSDL.
433 *      <li>If the <code>portName</code> port is not <code>null</code> and it
434 *      is not present in <code>serviceName</code> service in the WSDL.
435 *      <li>If the <code>wsdlDocumentLocation</code> is NOT <code>null</code>
436 *      and does not represent a valid WSDL.
437 *      <li>If the <code>wsdlDocumentLocation</code> is NOT <code>null</code> but
438 *      wsdl:wsdlLocation's namespace name cannot be got from the available
439 *      metadata.
440 *      </ul>
441 * @throws WebServiceException If an error occurs while creating the
442 *      <code>W3CEndpointReference</code>.
443 * @since JAX-WS 2.2
444 */
445 public W3CEndpointReference createW3CEndpointReference(String address,
446     QName interfaceName, QName serviceName, QName portName,
447     List<Element> metadata, String wsdlDocumentLocation, List<Element> referenceParameters,
448     List<Element> elements, Map<QName, String> attributes) {
449     throw new UnsupportedOperationException("JAX-WS 2.2 implementation must override this
450         default behaviour.");
451 }
452 /**
453 * Creates and publishes an endpoint object with the specified
454 * address, implementation object and web service features.
455 * <code>Provider</code> implementations must override the
456 * default implementation.
457 *
458 * @param address A URI specifying the address and transport/protocol
459 *      to use. A http: URI MUST result in the SOAP 1.1/HTTP
460 *      binding being used. Implementations may support other
461 *      URI schemes.
462 * @param implementor A service implementation object to which
463 *      incoming requests will be dispatched. The corresponding
464 *      class MUST be annotated with all the necessary Web service
465 *      annotations.
466 * @param features A list of WebServiceFeatures to configure on the
467 *      endpoint. Supported features not in the <code>features
468 *      </code> parameter will have their default values.
469 * @return The newly created endpoint.
470 * @since JAX-WS 2.2
471 */
472 public Endpoint createAndPublishEndpoint(String address,
473     Object implementor, WebServiceFeature ... features) {
474     throw new UnsupportedOperationException("JAX-WS 2.2 implementation must override this
475         default behaviour.");
476 }

```



```

477 /**
478  * Creates an endpoint object with the provided binding, implementation
479  * object and web service features. <code>Provider</code> implementations
480  * must override the default implementation.
481  *
482  * @param bindingId A URI specifying the desired binding (e.g. SOAP/HTTP)
483  * @param implementor A service implementation object to which
484  *      incoming requests will be dispatched. The corresponding
485  *      class MUST be annotated with all the necessary Web service
486  *      annotations.
487  * @param features A list of WebServiceFeatures to configure on the
488  *      endpoint. Supported features not in the <code>features
489  *      </code> parameter will have their default values.
490  * @return The newly created endpoint.
491  * @since JAX-WS 2.2
492  */
493 public Endpoint createEndpoint(String bindingId, Object implementor,
494     WebServiceFeature ... features) {
495     throw new UnsupportedOperationException("JAX-WS 2.2 implementation must override this
496         default behaviour.");
497 }
498 /**
499  * Creates an endpoint object with the provided binding, implementation
500  * class, invoker and web service features. Containers typically use
501  * this to create Endpoint objects. <code>Provider</code>
502  * implementations must override the default implementation.
503  *
504  * @param bindingId A URI specifying the desired binding (e.g. SOAP/HTTP).
505  *      Can be null.
506  * @param implementorClass A service implementation class that
507  *      MUST be annotated with all the necessary Web service
508  *      annotations.
509  * @param invoker that does the actual invocation on the service instance.
510  * @param features A list of WebServiceFeatures to configure on the
511  *      endpoint. Supported features not in the <code>features
512  *      </code> parameter will have their default values.
513  * @return The newly created endpoint.
514  * @since JAX-WS 2.2
515  */
516 public Endpoint createEndpoint(String bindingId, Class<?> implementorClass,
517     Invoker invoker, WebServiceFeature ... features) {
518     throw new UnsupportedOperationException("JAX-WS 2.2 implementation must override this
519         default behaviour.");
520 }
521 }

```

## 32.4 ServiceDelegate.java

Secuencia 310: javax/xml/ws/spi/ServiceDelegate.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.

```

```
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.spi;
27
28 import java.util.Iterator;
29 import javax.xml.namespace.QName;
30 import javax.xml.ws.Dispatch;
31 import javax.xml.ws.Service;
32 import javax.xml.ws.handler.HandlerResolver;
33 import javax.xml.ws.WebServiceFeature;
34 import javax.xml.bind.JAXBContext;
35 import javax.xml.ws.EndpointReference;
36 import javax.xml.ws.WebServiceException;
37
38
39 /**
40  * Service delegates are used internally by Service objects
41  * to allow pluggability of JAX-WS implementations.
42  * <p>
43  * Every Service object has its own delegate, created using
44  * the {@link javax.xml.ws.spi.Provider#createServiceDelegate} method. A Service
45  * object delegates all of its instance methods to its delegate.
46  *
47  * @see javax.xml.ws.Service
48  * @see javax.xml.ws.spi.Provider
49  *
50  * @since JAX-WS 2.0
51  */
52 public abstract class ServiceDelegate {
53
54     protected ServiceDelegate() {
55     }
```

```

56
57 /**
58  * The <code>getPort</code> method returns a proxy. A service client
59  * uses this proxy to invoke operations on the target
60  * service endpoint. The <code>serviceEndpointInterface</code>
61  * specifies the service endpoint interface that is supported by
62  * the created dynamic proxy instance.
63  *
64  * @param portName Qualified name of the service endpoint in
65  *                 the WSDL service description
66  * @param serviceEndpointInterface Service endpoint interface
67  *                                 supported by the dynamic proxy
68  * @return Object Proxy instance that
69  *         supports the specified service endpoint
70  *         interface
71  * @throws WebServiceException This exception is thrown in the
72  *         following cases:
73  *         <UL>
74  *         <LI>If there is an error in creation of
75  *         the proxy
76  *         <LI>If there is any missing WSDL metadata
77  *         as required by this method
78  *         <LI>If an illegal
79  *         <code>serviceEndpointInterface</code>
80  *         or <code>portName</code> is specified
81  *         </UL>
82  * @see java.lang.reflect.Proxy
83  * @see java.lang.reflect.InvocationHandler
84  */
85 public abstract <T> T getPort(QName portName,
86                             Class<T> serviceEndpointInterface);
87
88 /**
89  * The <code>getPort</code> method returns a proxy. A service client
90  * uses this proxy to invoke operations on the target
91  * service endpoint. The <code>serviceEndpointInterface</code>
92  * specifies the service endpoint interface that is supported by
93  * the created dynamic proxy instance.
94  *
95  * @param portName Qualified name of the service endpoint in
96  *                 the WSDL service description
97  * @param serviceEndpointInterface Service endpoint interface
98  *                                 supported by the dynamic proxy or instance
99  * @param features A list of WebServiceFeatures to configure on the
100  *                proxy. Supported features not in the <code>features
101  *                </code> parameter will have their default values.
102  * @return Object Proxy instance that
103  *         supports the specified service endpoint
104  *         interface
105  * @throws WebServiceException This exception is thrown in the
106  *         following cases:
107  *         <UL>
108  *         <LI>If there is an error in creation of

```

```

109      *           the proxy
110      *           <LI>If there is any missing WSDL metadata
111      *           as required by this method
112      *           <LI>If an illegal
113      *           <code>serviceEndpointInterface</code>
114      *           or <code>portName</code> is specified
115      *           <LI>If a feature is enabled that is not compatible
116      *           with this port or is unsupported.
117      *           </UL>
118      * @see java.lang.reflect.Proxy
119      * @see java.lang.reflect.InvocationHandler
120      * @see WebServiceFeature
121      *
122      * @since JAX-WS 2.1
123      **/
124      public abstract <T> T getPort(QName portName,
125          Class<T> serviceEndpointInterface, WebServiceFeature... features);
126
127      /**
128      * The <code>getPort</code> method returns a proxy.
129      * The parameter <code>endpointReference</code> specifies the
130      * endpoint that will be invoked by the returned proxy. If there
131      * are any reference parameters in the
132      * <code>endpointReference</code>, then those reference
133      * parameters MUST appear as SOAP headers, indicating them to be
134      * reference parameters, on all messages sent to the endpoint.
135      * The <code>endpointReference's</code> address MUST be used
136      * for invocations on the endpoint.
137      * The parameter <code>serviceEndpointInterface</code> specifies
138      * the service endpoint interface that is supported by the
139      * returned proxy.
140      * In the implementation of this method, the JAX-WS
141      * runtime system takes the responsibility of selecting a protocol
142      * binding (and a port) and configuring the proxy accordingly from
143      * the WSDL associated with this <code>Service</code> instance or
144      * from the metadata from the <code>endpointReference</code>.
145      * If this <code>Service</code> instance has a WSDL and
146      * the <code>endpointReference</code> metadata
147      * also has a WSDL, then the WSDL from this instance MUST be used.
148      * If this <code>Service</code> instance does not have a WSDL and
149      * the <code>endpointReference</code> does have a WSDL, then the
150      * WSDL from the <code>endpointReference</code> MAY be used.
151      * The returned proxy should not be reconfigured by the client.
152      * If this <code>Service</code> instance has a known proxy
153      * port that matches the information contained in
154      * the WSDL,
155      * then that proxy is returned, otherwise a WebServiceException
156      * is thrown.
157      * <p>
158      * Calling this method has the same behavior as the following
159      * <pre>
160      * <code>port = service.getPort(portName, serviceEndpointInterface);</code>
161      * </pre>

```

```

162 * where the <code>portName</code> is retrieved from the
163 * metadata of the <code>endpointReference</code> or from the
164 * <code>serviceEndpointInterface</code> and the WSDL
165 * associated with this <code>Service</code> instance.
166 *
167 * @param endpointReference The <code>EndpointReference</code>
168 * for the target service endpoint that will be invoked by the
169 * returned proxy.
170 * @param serviceEndpointInterface Service endpoint interface.
171 * @param features A list of <code>WebServiceFeatures</code> to configure on the
172 * proxy. Supported features not in the <code>features
173 * </code> parameter will have their default values.
174 * @return Object Proxy instance that supports the
175 * specified service endpoint interface.
176 * @throws WebServiceException
177 * <UL>
178 * <LI>If there is an error during creation
179 * of the proxy.
180 * <LI>If there is any missing WSDL metadata
181 * as required by this method.
182 * <LI>If the <code>endpointReference</code> metadata does
183 * not match the <code>serviceName</code> of this
184 * <code>Service</code> instance.
185 * <LI>If a <code>portName</code> cannot be extracted
186 * from the WSDL or <code>endpointReference</code> metadata.
187 * <LI>If an invalid
188 * <code>endpointReference</code>
189 * is specified.
190 * <LI>If an invalid
191 * <code>serviceEndpointInterface</code>
192 * is specified.
193 * <LI>If a feature is enabled that is not compatible
194 * with this port or is unsupported.
195 * </UL>
196 *
197 * @since JAX-WS 2.1
198 **/
199 public abstract <T> T getPort(EndpointReference endpointReference,
200 Class<T> serviceEndpointInterface, WebServiceFeature... features);
201
202
203 /**
204 * The <code>getPort</code> method returns a proxy. The parameter
205 * <code>serviceEndpointInterface</code> specifies the service
206 * endpoint interface that is supported by the returned proxy.
207 * In the implementation of this method, the JAX-WS
208 * runtime system takes the responsibility of selecting a protocol
209 * binding (and a port) and configuring the proxy accordingly.
210 * The returned proxy should not be reconfigured by the client.
211 *
212 * @param serviceEndpointInterface Service endpoint interface
213 * @return Object instance that supports the
214 * specified service endpoint interface

```

```

215     * @throws WebServiceException
216     *
217     *      <UL>
218     *          <LI>If there is an error during creation
219     *              of the proxy
220     *          <LI>If there is any missing WSDL metadata
221     *              as required by this method
222     *          <LI>If an illegal
223     *              <code>serviceEndpointInterface</code>
224     *              is specified
225     *      </UL>
226     **/
227     public abstract <T> T getPort(Class<T> serviceEndpointInterface);
228
229     /**
230     * The <code>getPort</code> method returns a proxy. The parameter
231     * <code>serviceEndpointInterface</code> specifies the service
232     * endpoint interface that is supported by the returned proxy.
233     * In the implementation of this method, the JAX-WS
234     * runtime system takes the responsibility of selecting a protocol
235     * binding (and a port) and configuring the proxy accordingly.
236     * The returned proxy should not be reconfigured by the client.
237     *
238     * @param serviceEndpointInterface Service endpoint interface
239     * @param features An array of <code>WebServiceFeatures</code> to configure on the
240     *      proxy. Supported features not in the <code>features
241     *      </code> parameter will have their default values.
242     * @return Object instance that supports the
243     *      specified service endpoint interface
244     * @throws WebServiceException
245     *
246     *      <UL>
247     *          <LI>If there is an error during creation
248     *              of the proxy
249     *          <LI>If there is any missing WSDL metadata
250     *              as required by this method
251     *          <LI>If an illegal
252     *              <code>serviceEndpointInterface</code>
253     *              is specified
254     *          <LI>If a feature is enabled that is not compatible
255     *              with this port or is unsupported.
256     *      </UL>
257     *
258     * @see WebServiceFeature
259     *
260     * @since JAX-WS 2.1
261     **/
262     public abstract <T> T getPort(Class<T> serviceEndpointInterface,
263     *      WebServiceFeature... features);
264
265     /**
266     * Creates a new port for the service. Ports created in this way contain
267     * no WSDL port type information and can only be used for creating

```

```

268     * <code>Dispatch</code>instances.
269     *
270     * @param portName Qualified name for the target service endpoint
271     * @param bindingId A URI identifier of a binding.
272     * @param endpointAddress Address of the target service endpoint as a URI
273     * @throws WebServiceException If any error in the creation of
274     * the port
275     *
276     * @see javax.xml.ws.soap.SOAPBinding#SOAP11HTTP_BINDING
277     * @see javax.xml.ws.soap.SOAPBinding#SOAP12HTTP_BINDING
278     * @see javax.xml.ws.http.HTTPBinding#HTTP_BINDING
279     **/
280     public abstract void addPort(QName portName, String bindingId,
281         String endpointAddress);
282
283
284
285     /**
286     * Creates a <code>Dispatch</code> instance for use with objects of
287     * the user's choosing.
288     *
289     * @param portName Qualified name for the target service endpoint
290     * @param type The class of object used for messages or message
291     * payloads. Implementations are required to support
292     * <code>javax.xml.transform.Source</code> and <code>javax.xml.soap.SOAPMessage</code>.
293     * @param mode Controls whether the created dispatch instance is message
294     * or payload oriented, i.e. whether the user will work with complete
295     * protocol messages or message payloads. E.g. when using the SOAP
296     * protocol, this parameter controls whether the user will work with
297     * SOAP messages or the contents of a SOAP body. Mode MUST be <code>MESSAGE</code>
298     * when type is <code>SOAPMessage</code>.
299     *
300     * @return Dispatch instance
301     * @throws WebServiceException If any error in the creation of
302     * the <code>Dispatch</code> object
303     * @see javax.xml.transform.Source
304     * @see javax.xml.soap.SOAPMessage
305     **/
306     public abstract <T> Dispatch<T> createDispatch(QName portName, Class<T> type,
307         Service.Mode mode);
308
309     /**
310     * Creates a <code>Dispatch</code> instance for use with objects of
311     * the user's choosing.
312     *
313     * @param portName Qualified name for the target service endpoint
314     * @param type The class of object used for messages or message
315     * payloads. Implementations are required to support
316     * <code>javax.xml.transform.Source</code> and <code>javax.xml.soap.SOAPMessage</code>.
317     * @param mode Controls whether the created dispatch instance is message
318     * or payload oriented, i.e. whether the user will work with complete
319     * protocol messages or message payloads. E.g. when using the SOAP
320     * protocol, this parameter controls whether the user will work with

```

```

321 * SOAP messages or the contents of a SOAP body. Mode MUST be <code>MESSAGE</code>
322 * when type is <code>SOAPMessage</code>.
323 * @param features A list of <code>WebServiceFeatures</code> to configure on the
324 * proxy. Supported features not in the <code>features
325 * </code> parameter will have their default values.
326 *
327 * @return Dispatch instance
328 * @throws WebServiceException If any error in the creation of
329 * the <code>Dispatch</code> object or if a
330 * feature is enabled that is not compatible with
331 * this port or is unsupported.
332 *
333 * @see javax.xml.transform.Source
334 * @see javax.xml.soap.SOAPMessage
335 * @see WebServiceFeature
336 *
337 * @since JAX-WS 2.1
338 **/
339 public abstract <T> Dispatch<T> createDispatch(QName portName, Class<T> type,
340 Service.Mode mode, WebServiceFeature... features);
341
342 /**
343 * Creates a <code>Dispatch</code> instance for use with objects of
344 * the user's choosing. If there
345 * are any reference parameters in the
346 * <code>endpointReference</code>, then those reference
347 * parameters MUST appear as SOAP headers, indicating them to be
348 * reference parameters, on all messages sent to the endpoint.
349 * The <code>endpointReference's</code> address MUST be used
350 * for invocations on the endpoint.
351 * In the implementation of this method, the JAX-WS
352 * runtime system takes the responsibility of selecting a protocol
353 * binding (and a port) and configuring the dispatch accordingly from
354 * the WSDL associated with this <code>Service</code> instance or
355 * from the metadata from the <code>endpointReference</code>.
356 * If this <code>Service</code> instance has a WSDL and
357 * the <code>endpointReference</code>
358 * also has a WSDL in its metadata, then the WSDL from this instance MUST be used.
359 * If this <code>Service</code> instance does not have a WSDL and
360 * the <code>endpointReference</code> does have a WSDL, then the
361 * WSDL from the <code>endpointReference</code> MAY be used.
362 * An implementation MUST be able to retrieve the <code>portName</code> from the
363 * <code>endpointReference</code> metadata.
364 * <p>
365 * This method behaves the same as calling
366 * <pre>
367 * <code>dispatch = service.createDispatch(portName, type, mode, features);</code>
368 * </pre>
369 * where the <code>portName</code> is retrieved from the
370 * WSDL or <code>EndpointReference</code> metadata.
371 *
372 * @param endpointReference The <code>EndpointReference</code>
373 * for the target service endpoint that will be invoked by the

```



```

374 * returned <code>Dispatch</code> object.
375 * @param type The class of object used to messages or message
376 * payloads. Implementations are required to support
377 * <code>javax.xml.transform.Source</code> and <code>javax.xml.soap.SOAPMessage</code>.
378 * @param mode Controls whether the created dispatch instance is message
379 * or payload oriented, i.e. whether the user will work with complete
380 * protocol messages or message payloads. E.g. when using the SOAP
381 * protocol, this parameter controls whether the user will work with
382 * SOAP messages or the contents of a SOAP body. Mode MUST be <code>MESSAGE</code>
383 * when type is <code>SOAPMessage</code>.
384 * @param features An array of <code>WebServiceFeatures</code> to configure on the
385 * proxy. Supported features not in the <code>features
386 * </code> parameter will have their default values.
387 *
388 * @return Dispatch instance
389 * @throws WebServiceException
390 *
391 * <UL>
392 * <LI>If there is any missing WSDL metadata
393 * as required by this method.
394 * <li>If the <code>endpointReference</code> metadata does
395 * not match the <code>serviceName</code> or <code>portName</code>
396 * of a WSDL associated
397 * with this <code>Service</code> instance.
398 * <li>If the <code>portName</code> cannot be determined
399 * from the <code>EndpointReference</code> metadata.
400 * <li>If any error in the creation of
401 * the <code>Dispatch</code> object.
402 * <li>If a feature is enabled that is not
403 * compatible with this port or is unsupported.
404 * </UL>
405 *
406 * @see javax.xml.transform.Source
407 * @see javax.xml.soap.SOAPMessage
408 * @see WebServiceFeature
409 *
410 * @since JAX-WS 2.1
411 */
412 public abstract <T> Dispatch<T> createDispatch(EndpointReference endpointReference,
413 Class<T> type, Service.Mode mode,
414 WebServiceFeature... features);
415
416 /**
417 * Creates a <code>Dispatch</code> instance for use with JAXB
418 * generated objects.
419 *
420 * @param portName Qualified name for the target service endpoint
421 * @param context The JAXB context used to marshall and unmarshall
422 * messages or message payloads.
423 * @param mode Controls whether the created dispatch instance is message
424 * or payload oriented, i.e. whether the user will work with complete
425 * protocol messages or message payloads. E.g. when using the SOAP

```

```

427     * protocol, this parameter controls whether the user will work with
428     * SOAP messages or the contents of a SOAP body.
429     *
430     * @return Dispatch instance
431     * @throws WebServiceException If any error in the creation of
432     *         the <code>Dispatch</code> object
433     *
434     * @see javax.xml.bind.JAXBContext
435     **/
436     public abstract Dispatch<Object> createDispatch(QName portName,
437         JAXBContext context, Service.Mode mode);
438
439
440     /**
441     * Creates a <code>Dispatch</code> instance for use with JAXB
442     * generated objects.
443     *
444     * @param portName Qualified name for the target service endpoint
445     * @param context The JAXB context used to marshall and unmarshall
446     * messages or message payloads.
447     * @param mode Controls whether the created dispatch instance is message
448     * or payload oriented, i.e. whether the user will work with complete
449     * protocol messages or message payloads. E.g. when using the SOAP
450     * protocol, this parameter controls whether the user will work with
451     * SOAP messages or the contents of a SOAP body.
452     * @param features A list of <code>WebServiceFeatures</code> to configure on the
453     * proxy. Supported features not in the <code>features
454     * </code> parameter will have their default values.
455     *
456     * @return Dispatch instance
457     * @throws WebServiceException If any error in the creation of
458     *         the <code>Dispatch</code> object or if a
459     *         feature is enabled that is not compatible with
460     *         this port or is unsupported.
461     *
462     * @see javax.xml.bind.JAXBContext
463     * @see WebServiceFeature
464     *
465     * @since JAX-WS 2.1
466     **/
467     public abstract Dispatch<Object> createDispatch(QName portName,
468         JAXBContext context, Service.Mode mode, WebServiceFeature... features);
469
470     /**
471     * Creates a <code>Dispatch</code> instance for use with JAXB
472     * generated objects. If there
473     * are any reference parameters in the
474     * <code>endpointReference</code>, then those reference
475     * parameters MUST appear as SOAP headers, indicating them to be
476     * reference parameters, on all messages sent to the endpoint.
477     * The <code>endpointReference's</code> address MUST be used
478     * for invocations on the endpoint.
479     * In the implementation of this method, the JAX-WS

```

```

480 * runtime system takes the responsibility of selecting a protocol
481 * binding (and a port) and configuring the dispatch accordingly from
482 * the WSDL associated with this Service instance or
483 * from the metadata from the endpointReference.
484 * If this Service instance has a WSDL and
485 * the endpointReference
486 * also has a WSDL in its metadata, then the WSDL from this instance
487 * MUST be used.
488 * If this Service instance does not have a WSDL and
489 * the endpointReference does have a WSDL, then the
490 * WSDL from the endpointReference MAY be used.
491 * An implementation MUST be able to retrieve the portName from the
492 * endpointReference metadata.
493 * <p>
494 * This method behaves the same as calling
495 * <pre>
496 * dispatch = service.createDispatch(portName, context, mode, features);
497 * </pre>
498 * where the portName is retrieved from the
499 * WSDL or endpointReference metadata.
500 *
501 * @param endpointReference The EndpointReference
502 * for the target service endpoint that will be invoked by the
503 * returned Dispatch object.
504 * @param context The JAXB context used to marshal and unmarshal
505 * messages or message payloads.
506 * @param mode Controls whether the created dispatch instance is message
507 * or payload oriented, i.e. whether the user will work with complete
508 * protocol messages or message payloads. E.g. when using the SOAP
509 * protocol, this parameter controls whether the user will work with
510 * SOAP messages or the contents of a SOAP body.
511 * @param features An array of WebServiceFeatures to configure on the
512 * proxy. Supported features not in the features
513 * parameter will have their default values.
514 *
515 * @return Dispatch instance
516 * @throws WebServiceException
517 * <UL>
518 * <li>If there is any missing WSDL metadata
519 * as required by this method.
520 * <li>If the endpointReference metadata does
521 * not match the serviceName or portName
522 * of a WSDL associated
523 * with this Service instance.
524 * <li>If the portName cannot be determined
525 * from the EndpointReference metadata.
526 * <li>If any error in the creation of
527 * the Dispatch object.
528 * <li>if a feature is enabled that is not
529 * compatible with this port or is unsupported.
530 * </UL>
531 *
532 * @see javax.xml.bind.JAXBContext

```

```
533     * @see WebServiceFeature
534     *
535     * @since JAX-WS 2.1
536     **/
537     public abstract Dispatch<Object> createDispatch(EndpointReference endpointReference,
538             JAXBContext context, Service.Mode mode,
539             WebServiceFeature... features);
540
541
542     /**
543     * Gets the name of this service.
544     * @return Qualified name of this service
545     **/
546     public abstract QName getServiceName();
547
548     /**
549     * Returns an <code>Iterator</code> for the list of
550     * <code>QName</code>s of service endpoints grouped by this
551     * service
552     *
553     * @return Returns <code>java.util.Iterator</code> with elements
554     *         of type <code>javax.xml.namespace.QName</code>
555     * @throws WebServiceException If this Service class does not
556     *         have access to the required WSDL metadata
557     **/
558     public abstract Iterator<javax.xml.namespace.QName> getPorts();
559
560     /**
561     * Gets the location of the WSDL document for this Service.
562     *
563     * @return URL for the location of the WSDL document for
564     *         this service
565     **/
566     public abstract java.net.URL getWSDLDocumentLocation();
567
568     /**
569     * Returns the configured handler resolver.
570     *
571     * @return HandlerResolver The <code>HandlerResolver</code> being
572     *         used by this <code>Service</code> instance, or <code>null</code>
573     *         if there isn't one.
574     **/
575     public abstract HandlerResolver getHandlerResolver();
576
577     /**
578     * Sets the <code>HandlerResolver</code> for this <code>Service</code>
579     * instance.
580     * <p>
581     * The handler resolver, if present, will be called once for each
582     * proxy or dispatch instance that is created, and the handler chain
583     * returned by the resolver will be set on the instance.
584     *
585     * @param handlerResolver The <code>HandlerResolver</code> to use
```

```

586      *      for all subsequently created proxy/dispatch objects.
587      *
588      * @see javax.xml.ws.handler.HandlerResolver
589      **/
590      public abstract void setHandlerResolver(HandlerResolver handlerResolver);
591
592      /**
593      * Returns the executor for this <code>Service</code>instance.
594      *
595      * The executor is used for all asynchronous invocations that
596      * require callbacks.
597      *
598      * @return The <code>java.util.concurrent.Executor</code> to be
599      *         used to invoke a callback.
600      *
601      * @see java.util.concurrent.Executor
602      **/
603      public abstract java.util.concurrent.Executor getExecutor();
604
605      /**
606      * Sets the executor for this <code>Service</code> instance.
607      *
608      * The executor is used for all asynchronous invocations that
609      * require callbacks.
610      *
611      * @param executor The <code>java.util.concurrent.Executor</code>
612      *                 to be used to invoke a callback.
613      *
614      * @throws SecurityException If the instance does not support
615      *                 setting an executor for security reasons (e.g. the
616      *                 necessary permissions are missing).
617      *
618      * @see java.util.concurrent.Executor
619      **/
620      public abstract void setExecutor(java.util.concurrent.Executor executor);
621
622  }

```

## 32.5 WebServiceFeatureAnnotation.java

Secuencia 311: javax/xml/ws/spi/WebServiceFeatureAnnotation.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws.spi;
27
28 import java.lang.annotation.Documented;
29 import java.lang.annotation.Target;
30 import java.lang.annotation.ElementType;
31 import java.lang.annotation.Retention;
32 import java.lang.annotation.RetentionPolicy;
33 import javax.xml.ws.WebServiceFeature;
34 import javax.xml.ws.WebServiceRef;
35 import javax.xml.ws.WebServiceRefs;
36 import javax.xml.ws.RespectBinding;
37 import javax.xml.ws.soap.Addressing;
38 import javax.xml.ws.soap.MTOM;
39
40 /**
41  * Annotation used to identify other annotations
42  * as a WebServiceFeature.
43  * <p>
44  * Each WebServiceFeature annotation annotated with
45  * this annotation MUST contain an
46  * enabled property of type
47  * boolean with a default value of true.
48  * <p>
49  * JAX-WS defines the following
50  * WebServiceFeature annotations (Addressing,
51  * MTOM, RespectBinding), however, an implementation
52  * may define vendors specific annotations for other features.
53  * <p>
54  * Annotations annotated with WebServiceFeatureAnnotation MUST
55  * have the same @Target of {@link WebServiceRef} annotation, so that the resulting
56  * feature annotation can be used in conjunction with the {@link WebServiceRef}
57  * annotation if necessary.
58  * <p>
59  * If a JAX-WS implementation encounters an annotation annotated
60  * with the WebServiceFeatureAnnotation that it does not
61  * recognize/support an error MUST be given.
62  * <p>
63  *
64  * @see Addressing
65  * @see MTOM
```

```

66  * @see RespectBinding
67  *
68  * @since JAX-WS 2.1
69  */
70  @Target(ElementType.ANNOTATION_TYPE)
71  @Retention(RetentionPolicy.RUNTIME)
72  @Documented
73  public @interface WebServiceFeatureAnnotation {
74      /**
75       * Unique identifier for the WebServiceFeature. This
76       * identifier MUST be unique across all implementations
77       * of JAX-WS.
78       */
79      String id();
80
81      /**
82       * The <code>WebServiceFeature</code> bean that is associated
83       * with the <code>WebServiceFeature</code> annotation
84       */
85      Class<? extends WebServiceFeature> bean();
86  }

```

### 33 Xml Ws Spi Http (javax.xml.ws.spi.http package)

#### 33.1 HttpContext.java

Secuencia 312: javax/xml/ws/spi/http/HttpContext.java

```

1  /*
2  * Copyright (c) 2009, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.spi.http;

```

```
27
28 import javax.xml.ws.Endpoint;
29 import java.util.Set;
30
31 /**
32  * HttpContext represents a mapping between the root URI path of a web
33  * service to a {@link HttpHandler} which is invoked to handle requests
34  * destined for that path on the associated container.
35  * <p>
36  * Container provides the implementation for this and it matches
37  * web service requests to corresponding HttpContext objects.
38  *
39  * @author Jitendra Kotamraju
40  * @since JAX-WS 2.2
41  */
42 public abstract class HttpContext {
43
44     protected HttpHandler handler;
45
46     /**
47      * JAX-WS runtime sets its handler during
48      * {@link Endpoint#publish(HttpContext)} to handle
49      * HTTP requests for this context. Container or its extensions
50      * use this handler to process the requests.
51      *
52      * @param handler the handler to set for this context
53      */
54     public void setHandler(HttpHandler handler) {
55         this.handler = handler;
56     }
57
58     /**
59      * Returns the path for this context. This path uniquely identifies
60      * an endpoint inside an application and the path is relative to
61      * application's context path. Container should give this
62      * path based on how it matches request URIs to this HttpContext object.
63      *
64      * <p>
65      * For servlet container, this is typically a url-pattern for an endpoint.
66      *
67      * <p>
68      * Endpoint's address for this context can be computed as follows:
69      * <pre>
70      *   HttpExchange exch = ...;
71      *   String endpointAddress =
72      *       exch.getScheme() + "://"
73      *       + exch.getLocalAddress().getHostName()
74      *       + ":" + exch.getLocalAddress().getPort()
75      *       + exch.getContextPath() + getPath();
76      * </pre>
77      *
78      * @return this context's path
79      */

```



```

80     public abstract String getPath();
81
82     /**
83      * Returns an attribute value for container's configuration
84      * and other data that can be used by jax-ws runtime.
85      *
86      * @param name attribute name
87      * @return attribute value
88      */
89     public abstract Object getAttribute(String name);
90
91     /**
92      * Returns all attribute names for container's configuration
93      * and other data that can be used by jax-ws runtime.
94      *
95      * @return set of all attribute names
96      */
97     public abstract Set<String> getAttributeNames();
98
99 }

```

### 33.2 HttpExchange.java

Secuencia 313: javax/xml/ws/spi/http/HttpExchange.java

```

1  /*
2   * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.ws.spi.http;
27
28 import javax.xml.ws.handler.MessageContext;
29 import java.io.InputStream;

```

```
30 import java.io.OutputStream;
31 import java.io.IOException;
32 import java.net.InetSocketAddress;
33 import java.util.List;
34 import java.util.Map;
35 import java.util.Set;
36 import java.security.Principal;
37
38 /**
39  * This class encapsulates a HTTP request received and a
40  * response to be generated in one exchange. It provides methods
41  * for examining the request from the client, and for building and
42  * sending the response.
43  * <p>
44  * A <code>HttpExchange</code> must be closed to free or reuse
45  * underlying resources. The effect of failing to close an exchange
46  * is undefined.
47  *
48  * @author Jitendra Kotamraju
49  * @since JAX-WS 2.2
50  */
51 public abstract class HttpExchange {
52
53     /**
54      * Standard property: cipher suite value when the request is received
55      * over HTTPS
56      * <p>Type: String
57      */
58     public static final String REQUEST_CIPHER_SUITE =
59         "javax.xml.ws.spi.http.request.cipher.suite";
60
61     /**
62      * Standard property: bit size of the algorithm when the request is
63      * received over HTTPS
64      * <p>Type: Integer
65      */
66     public static final String REQUEST_KEY_SIZE =
67         "javax.xml.ws.spi.http.request.key.size";
68
69     /**
70      * Standard property: A SSL certificate, if any, associated with the request
71      *
72      * <p>Type: java.security.cert.X509Certificate[]
73      * The order of this array is defined as being in ascending order of trust.
74      * The first certificate in the chain is the one set by the client, the next
75      * is the one used to authenticate the first, and so on.
76      */
77     public static final String REQUEST_X509CERTIFICATE =
78         "javax.xml.ws.spi.http.request.cert.X509Certificate";
79
80     /**
81      * Returns an immutable Map containing the HTTP headers that were
82      * included with this request. The keys in this Map will be the header
```

```
83      * names, while the values will be a List of Strings containing each value
84      * that was included (either for a header that was listed several times,
85      * or one that accepts a comma-delimited list of values on a single line).
86      * In either of these cases, the values for the header name will be
87      * presented in the order that they were included in the request.
88      * <p>
89      * The keys in Map are case-insensitive.
90      *
91      * @return an immutable Map which can be used to access request headers
92      */
93      public abstract Map<String, List<String>> getRequestHeaders();
94
95      /**
96      * Returns the value of the specified request header. If the request
97      * did not include a header of the specified name, this method returns
98      * null. If there are multiple headers with the same name, this method
99      * returns the first header in the request. The header name is
100     * case-insensitive. This is a convenience method to get a header
101     * (instead of using the {@link #getRequestHeaders}).
102     *
103     * @param name the name of the request header
104     * @return returns the value of the requested header,
105     *         or null if the request does not have a header of that name
106     */
107     public abstract String getRequestHeader(String name);
108
109     /**
110     * Returns a mutable Map into which the HTTP response headers can be stored
111     * and which will be transmitted as part of this response. The keys in the
112     * Map will be the header names, while the values must be a List of Strings
113     * containing each value that should be included multiple times
114     * (in the order that they should be included).
115     * <p>
116     * The keys in Map are case-insensitive.
117     *
118     * @return a mutable Map which can be used to set response headers.
119     */
120     public abstract Map<String, List<String>> getResponseHeaders();
121
122     /**
123     * Adds a response header with the given name and value. This method
124     * allows a response header to have multiple values. This is a
125     * convenience method to add a response header (instead of using the
126     * {@link #getResponseHeaders}).
127     *
128     * @param name the name of the header
129     * @param value the additional header value. If it contains octet string,
130     *             it should be encoded according to
131     *             RFC 2047 (http://www.ietf.org/rfc/rfc2047.txt)
132     *
133     * @see #getResponseHeaders
134     */
135     public abstract void addResponseHeader(String name, String value);
```

```
136
137 /**
138  * Returns the part of the request's URI from the protocol
139  * name up to the query string in the first line of the HTTP request.
140  * Container doesn't decode this string.
141  *
142  * @return the request URI
143  */
144 public abstract String getRequestURI();
145
146 /**
147  * Returns the context path of all the endpoints in an application.
148  * This path is the portion of the request URI that indicates the
149  * context of the request. The context path always comes first in a
150  * request URI. The path starts with a "/" character but does not
151  * end with a "/" character. If this method returns "", the request
152  * is for default context. The container does not decode this string.
153  *
154  * <p>
155  * Context path is used in computing the endpoint address. See
156  * {@link HttpContext#getPath}
157  *
158  * @return context path of all the endpoints in an application
159  * @see HttpContext#getPath
160  */
161 public abstract String getContextPath();
162
163 /**
164  * Get the HTTP request method
165  *
166  * @return the request method
167  */
168 public abstract String getRequestMethod();
169
170 /**
171  * Returns a {@link HttpContext} for this exchange.
172  * Container matches the request with the associated Endpoint's HttpContext
173  *
174  * @return the HttpContext for this exchange
175  */
176 public abstract HttpContext getHttpContext();
177
178 /**
179  * This must be called to end an exchange. Container takes care of
180  * closing request and response streams. This must be called so that
181  * the container can free or reuse underlying resources.
182  *
183  * @throws IOException if any i/o error
184  */
185 public abstract void close() throws IOException;
186
187 /**
188  * Returns a stream from which the request body can be read.
```

```
189     * Multiple calls to this method will return the same stream.
190     *
191     * @return the stream from which the request body can be read.
192     * @throws IOException if any i/o error during request processing
193     */
194     public abstract InputStream getRequestBody() throws IOException;
195
196     /**
197     * Returns a stream to which the response body must be
198     * written. {@link #setStatus()} must be called prior to calling
199     * this method. Multiple calls to this method (for the same exchange)
200     * will return the same stream.
201     *
202     * @return the stream to which the response body is written
203     * @throws IOException if any i/o error during response processing
204     */
205     public abstract OutputStream getResponseBody() throws IOException;
206
207     /**
208     * Sets the HTTP status code for the response.
209     *
210     * <p>
211     * This method must be called prior to calling {@link #getResponseBody}.
212     *
213     * @param status the response code to send
214     * @see #getResponseBody
215     */
216     public abstract void setStatus(int status);
217
218     /**
219     * Returns the unresolved address of the remote entity invoking
220     * this request.
221     *
222     * @return the InetAddress of the caller
223     */
224     public abstract InetAddress getRemoteAddress();
225
226     /**
227     * Returns the unresolved local address on which the request was received.
228     *
229     * @return the InetAddress of the local interface
230     */
231     public abstract InetAddress getLocalAddress();
232
233     /**
234     * Returns the protocol string from the request in the form
235     * <i>protocol/majorVersion.minorVersion</i>. For example,
236     * "HTTP/1.1"
237     *
238     * @return the protocol string from the request
239     */
240     public abstract String getProtocol();
241
```

```

242  /**
243   * Returns the name of the scheme used to make this request,
244   * for example: http, or https.
245   *
246   * @return name of the scheme used to make this request
247   */
248  public abstract String getScheme();
249
250  /**
251   * Returns the extra path information that follows the web service
252   * path but precedes the query string in the request URI and will start
253   * with a "/" character.
254   *
255   * <p>
256   * This can be used for {@link MessageContext#PATH_INFO}
257   *
258   * @return decoded extra path information of web service.
259   *         It is the path that comes
260   *         after the web service path but before the query string in the
261   *         request URI
262   *         <tt>null</tt> if there is no extra path in the request URI
263   */
264  public abstract String getPathInfo();
265
266  /**
267   * Returns the query string that is contained in the request URI
268   * after the path.
269   *
270   * <p>
271   * This can be used for {@link MessageContext#QUERY_STRING}
272   *
273   * @return undecoded query string of request URI, or
274   *         <tt>null</tt> if the request URI doesn't have one
275   */
276  public abstract String getQueryString();
277
278  /**
279   * Returns an attribute that is associated with this
280   * <code>HttpExchange</code>. JAX-WS handlers and endpoints may then
281   * access the attribute via {@link MessageContext}.
282   * <p>
283   * Servlet containers must expose {@link MessageContext#SERVLET_CONTEXT},
284   * {@link MessageContext#SERVLET_REQUEST}, and
285   * {@link MessageContext#SERVLET_RESPONSE}
286   * as attributes.
287   *
288   * <p>If the request has been received by the container using HTTPS, the
289   * following information must be exposed as attributes. These attributes
290   * are {@link #REQUEST_CIPHER_SUITE}, and {@link #REQUEST_KEY_SIZE}.
291   * If there is a SSL certificate associated with the request, it must
292   * be exposed using {@link #REQUEST_X509CERTIFICATE}
293   *
294   * @param name attribute name

```

```

295     * @return the attribute value, or <tt>null</tt> if the attribute doesn't
296     *         exist
297     */
298     public abstract Object getAttribute(String name);
299
300     /**
301     * Gives all the attribute names that are associated with
302     * this <code>HttpExchange</code>.
303     *
304     * @return set of all attribute names
305     * @see #getAttribute(String)
306     */
307     public abstract Set<String> getAttributeNames();
308
309     /**
310     * Returns the {@link Principal} that represents the authenticated
311     * user for this <code>HttpExchange</code>.
312     *
313     * @return Principal for an authenticated user, or
314     *         <tt>null</tt> if not authenticated
315     */
316     public abstract Principal getUserPrincipal();
317
318     /**
319     * Indicates whether an authenticated user is included in the specified
320     * logical "role".
321     *
322     * @param role specifies the name of the role
323     * @return <tt>true</tt> if the user making this request belongs to a
324     *         given role
325     */
326     public abstract boolean isUserInRole(String role);
327
328 }

```

### 33.3 HttpHandler.java

Secuencia 314: javax/xml/ws/spi/http/HttpHandler.java

```

1  /*
2  * Copyright (c) 2009, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.ws.spi.http;
27
28  import javax.xml.ws.Endpoint;
29  import java.io.IOException;
30
31  /**
32   * A handler which is invoked to process HTTP requests.
33   * <p>
34   * JAX-WS runtime provides the implementation for this and sets
35   * it using {@link HttpContext#setHandler(Handler)} during
36   * {@link Endpoint#publish(HttpContext) }
37   *
38   * @author Jitendra Kotamraju
39   * @since JAX-WS 2.2
40   */
41  public abstract class Handler {
42      /**
43       * Handles a given request and generates an appropriate response.
44       * See {@link HttpExchange} for a description of the steps
45       * involved in handling an exchange. Container invokes this method
46       * when it receives an incoming request.
47       *
48       * @param exchange the exchange containing the request from the
49       *     client and used to send the response
50       * @throws IOException when an I/O error happens during request
51       *     handling
52       */
53      public abstract void handle(HttpExchange exchange) throws IOException;
54  }

```

### 33.4 package-info.java

Secuencia 315: javax/xml/ws/spi/http/package-info.java

```

1  /*
2   * Copyright (c) 2009, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  */

```



```

11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 /**
27  Provides HTTP SPI that is used for portable deployment of JAX-WS
28  web services in containers(for e.g. servlet containers). This SPI
29  is not for end developers but provides a way for the container
30  developers to deploy JAX-WS services portably.
31
32  <p>
33  The portable deployment is done as below:
34  <ol>
35  <li>Container creates {@link javax.xml.ws.Endpoint} objects for an
36  application. The necessary information to create Endpoint objects
37  may be got from web service deployment descriptor files.</li>
38  <li>Container needs to create {@link javax.xml.ws.spi.http.HttpContext}
39  objects for the deployment. For example, a HttpContext could be
40  created using servlet configuration(for e.g url-pattern) for the
41  web service in servlet container case.</li>
42  <li>Then publishes all the endpoints using
43  {@link javax.xml.ws.Endpoint#publish(HttpContext)}. During publish(),
44  JAX-WS runtime registers a {@link javax.xml.ws.spi.http.HttpHandler}
45  callback to handle incoming requests or
46  {@link javax.xml.ws.spi.http.HttpExchange} objects. The HttpExchange
47  object encapsulates a HTTP request and a response.
48  </ol>
49
50  <pre>
51  Container                                JAX-WS runtime
52  -----                                -
53  1. Creates Invoker1, ... InvokerN
54  2. Provider.createEndpoint(...)  --> 3. creates Endpoint1
55     configures Endpoint1
56     ...
57  4. Provider.createEndpoint(...)  --> 5. creates EndpointN
58     configures EndpointN
59  6. Creates ApplicationContext
60  7. creates HttpContext1, ... HttpContextN
61  8. Endpoint1.publish(HttpContext1) --> 9. creates HttpHandler1
62                                           HttpContext1.setHandler(HttpHandler1)
63  ...

```

```

64 10. EndpointN.publish(HttpContextN) --> 11. creates HttpHandlerN
65      HttpContextN.setHandler(HttpHandlerN)
66
67 </pre>
68
69 The request processing is done as below(for every request):
70 <pre>
71 Container                                JAX-WS runtime
72 -----                                -----
73 1. Creates a HttpExchange
74 2. Gets handler from HttpContext
75 3. HttpHandler.handle(HttpExchange) --> 4. reads request from HttpExchange
76                                     <-- 5. Calls Invoker
77 6. Invokes the actual instance
78                                     7. Writes the response to HttpExchange
79 </pre>
80
81 <p>
82 The portable undeployment is done as below:
83 <pre>
84 Container
85 -----
86 1. @preDestroy on instances
87 2. Endpoint1.stop()
88 ...
89 3. EndpointN.stop()
90 </pre>
91
92 @author Jitendra Kotamraju
93 @since JAX-WS 2.2
94 */
95 package javax.xml.ws.spi.http;

```

## 34 Xml Ws Wsaddressing (javax.xml.ws.wsaddressing package)

### 34.1 W3CEndpointReference.java

Secuencia 316: javax/xml/ws/wsaddressing/W3CEndpointReference.java

```

1  /*
2  * Copyright (c) 2005, 2014, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.ws.wsaddressing;
27
28
29  import org.w3c.dom.Element;
30
31  import javax.xml.bind.JAXBContext;
32  import javax.xml.bind.JAXBException;
33  import javax.xml.bind.Marshaller;
34  import javax.xml.bind.annotation.XmlAnyAttribute;
35  import javax.xml.bind.annotation.XmlAnyElement;
36  import javax.xml.bind.annotation.XmlElement;
37  import javax.xml.bind.annotation.XmlRootElement;
38  import javax.xml.bind.annotation.XmlType;
39  import javax.xml.bind.annotation.XmlValue;
40  import javax.xml.namespace.QName;
41  import javax.xml.transform.Result;
42  import javax.xml.transform.Source;
43  import javax.xml.ws.EndpointReference;
44  import javax.xml.ws.WebServiceException;
45  import java.util.List;
46  import java.util.Map;
47
48
49  /**
50   * This class represents a W3C Addressing EndpointReference which is
51   * a remote reference to a web service endpoint that supports the
52   * W3C WS-Addressing 1.0 - Core Recommendation.
53   * <p>
54   * Developers should use this class in their SEIs if they want to
55   * pass/return endpoint references that represent the W3C WS-Addressing
56   * recommendation.
57   * <p>
58   * JAXB will use the JAXB annotations and bind this class to XML infoset
59   * that is consistent with that defined by WS-Addressing. See
60   * <a href="http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/">
61   * WS-Addressing</a>
62   * for more information on WS-Addressing EndpointReferences.
63   *
64   * @since JAX-WS 2.1
65   */
66
67  // XmlRootElement allows this class to be marshalled on its own
68  @XmlRootElement(name="EndpointReference",namespace=W3CEndpointReference.NS)
```

```

69 @XmlType(name="EndpointReferenceType",namespace=W3CEndpointReference.NS)
70 public final class W3CEndpointReference extends EndpointReference {
71
72     private final JAXBContext w3cjc = getW3CJaxbContext();
73
74     // should be changed to package private, keeping original modifier to keep backwards
75     // compatibility
76     protected static final String NS = "http://www.w3.org/2005/08/addressing";
77
78     // default constructor forbidden ...
79     // should be private, keeping original modifier to keep backwards compatibility
80     protected W3CEndpointReference() {
81     }
82
83     /**
84      * Creates an EPR from infoaset representation
85      *
86      * @param source A source object containing valid XmlInfoaset
87      * instance consistent with the W3C WS-Addressing Core
88      * recommendation.
89      *
90      * @throws WebServiceException
91      * If the source does NOT contain a valid W3C WS-Addressing
92      * EndpointReference.
93      *
94      * @throws NullPointerException
95      * If the <code>null</code> <code>source</code> value is given
96      */
97     public W3CEndpointReference(Source source) {
98         try {
99             W3CEndpointReference epr = w3cjc.createUnmarshaller().unmarshal(source,
100                 W3CEndpointReference.class).getValue();
101             this.address = epr.address;
102             this.metadata = epr.metadata;
103             this.referenceParameters = epr.referenceParameters;
104             this.elements = epr.elements;
105             this.attributes = epr.attributes;
106         } catch (JAXBException e) {
107             throw new WebServiceException("Error unmarshalling W3CEndpointReference ", e);
108         } catch (ClassCastException e) {
109             throw new WebServiceException("Source did not contain W3CEndpointReference", e);
110         }
111     }
112
113     /**
114      * {@inheritDoc}
115      */
116     public void writeTo(Result result){
117         try {
118             Marshaller marshaller = w3cjc.createMarshaller();
119             marshaller.marshal(this, result);
120         } catch (JAXBException e) {
121             throw new WebServiceException("Error marshalling W3CEndpointReference. ", e);
122         }
123     }

```

```

120     }
121
122     private static JAXBContext getW3CJaxbContext() {
123         try {
124             return JAXBContext.newInstance(W3CEndpointReference.class);
125         } catch (JAXBException e) {
126             throw new WebServiceException("Error creating JAXBContext for W3CEndpointReference. ",
127                 e);
128         }
129
130         // private but necessary properties for databinding
131         @XmlElement(name="Address",namespace=NS)
132         private Address address;
133         @XmlElement(name="ReferenceParameters",namespace=NS)
134         private Elements referenceParameters;
135         @XmlElement(name="Metadata",namespace=NS)
136         private Elements metadata;
137         // attributes and elements are not private for performance reasons
138         // (JAXB can bypass reflection)
139         @XmlAnyAttribute
140         Map<QName,String> attributes;
141         @XmlAnyElement
142         List<Element> elements;
143
144
145         @XmlType(name="address", namespace=W3CEndpointReference.NS)
146         private static class Address {
147             protected Address() {}
148             @XmlValue
149             String uri;
150             @XmlAnyAttribute
151             Map<QName,String> attributes;
152         }
153
154
155         @XmlType(name="elements", namespace=W3CEndpointReference.NS)
156         private static class Elements {
157             protected Elements() {}
158             @XmlAnyElement
159             List<Element> elements;
160             @XmlAnyAttribute
161             Map<QName,String> attributes;
162         }
163
164     }

```

## 34.2 W3CEndpointReferenceBuilder.java

Secuencia 317: javax/xml/ws/wsaddressing/W3CEndpointReferenceBuilder.java

```

1  /*
2  * Copyright (c) 2005, 2011, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.ws.wsaddressing;
27
28
29 import org.w3c.dom.Element;
30
31 import java.util.ArrayList;
32 import java.util.List;
33 import java.util.Map;
34 import java.util.HashMap;
35 import javax.xml.namespace.QName;
36 import javax.xml.ws.WebServiceException;
37 import javax.xml.ws.spi.Provider;
38
39
40 /**
41  * This class is used to build <code>W3CEndpointReference</code>
42  * instances. The intended use of this class is for
43  * an application component, for example a factory component,
44  * to create an <code>W3CEndpointReference</code> for a
45  * web service endpoint published by the same
46  * Java EE application. It can also be used to create
47  * <code>W3CEndpointReferences</code> for an Java SE based
48  * endpoint by providing the <code>address</code> property.
49  * <p>
50  * When creating a <code>W3CEndpointReference</code> for an
51  * endpoint that is not published by the same Java EE application,
52  * the <code>address</code> property MUST be specified.
53  * <p>
54  * When creating a <code>W3CEndpointReference</code> for an endpoint
55  * published by the same Java EE application, the <code>address</code>
56  * property MAY be <code>null</code> but then the <code>serviceName</code>
```

```

57 * and <code>endpointName</code> MUST specify an endpoint published by
58 * the same Java EE application.
59 * <p>
60 * When the <code>wsdlDocumentLocation</code> is specified it MUST refer
61 * to a valid WSDL document and the <code>serviceName</code> and
62 * <code>endpointName</code> (if specified) MUST match a service and port
63 * in the WSDL document.
64 *
65 * @since JAX-WS 2.1
66 */
67 public final class W3CEndpointReferenceBuilder {
68     /**
69     * Creates a new <code>W3CEndpointReferenceBuilder</code> instance.
70     */
71     public W3CEndpointReferenceBuilder() {
72         referenceParameters = new ArrayList<Element>();
73         metadata = new ArrayList<Element>();
74         attributes = new HashMap<QName, String>();
75         elements = new ArrayList<Element>();
76     }
77
78     /**
79     * Sets the <code>address</code> to the
80     * <code>W3CEndpointReference</code> instance's
81     * <code>wsa:Address</code>.
82     * <p>
83     * The <code>address</code> MUST be set to a non-<code>null</code>
84     * value when building a <code>W3CEndpointReference</code> for a
85     * web service endpoint that is not published by the same
86     * Java EE application or when running on Java SE.
87     *
88     * @param address The address of the endpoint to be targeted
89     *     by the returned <code>W3CEndpointReference</code>.
90     *
91     * @return A <code>W3CEndpointReferenceBuilder</code> instance with
92     *     the <code>address</code> set to the <code>wsa:Address</code>.
93     */
94     public W3CEndpointReferenceBuilder address(String address) {
95         this.address = address;
96         return this;
97     }
98
99     /**
100     * Sets the <code>interfaceName</code> as the
101     * <code>wsam:InterfaceName</code> element in the
102     * <code>wsa:Metadata</code> element.
103     *
104     * See <a href="http://www.w3.org/TR/2007/REC-ws-addr-metadata-20070904/#refmetadatfromepr">
105     * 2.1 Referencing WSDL Metadata from an EPR</a> for more details.
106     *
107     * @param interfaceName The port type name of the endpoint to be targeted
108     *     by the returned <code>W3CEndpointReference</code>.
109     *

```

```

110     * @return A <code>W3CEndpointReferenceBuilder</code> instance with
111     * the <code>interfaceName</code> as <code>wsam:InterfaceName</code>
112     * element added to the <code>wsa:Metadata</code> element
113     */
114     public W3CEndpointReferenceBuilder interfaceName(QName interfaceName) {
115         this.interfaceName = interfaceName;
116         return this;
117     }
118
119     /**
120     * Sets the <code>serviceName</code> as the
121     * <code>wsam:ServiceName</code> element in the
122     * <code>wsa:Metadata</code> element.
123     *
124     * See <a href="http://www.w3.org/TR/2007/REC-ws-addr-metadata-20070904/#refmetadatfromepr">
125     * 2.1 Referencing WSDL Metadata from an EPR</a> for more details.
126     *
127     * @param serviceName The service name of the endpoint to be targeted
128     * by the returned <code>W3CEndpointReference</code>. This property
129     * may also be used with the <code>endpointName</code> (portName)
130     * property to lookup the <code>address</code> of a web service
131     * endpoint that is published by the same Java EE application.
132     *
133     * @return A <code>W3CEndpointReferenceBuilder</code> instance with
134     * the <code>serviceName</code> as <code>wsam:ServiceName</code>
135     * element added to the <code>wsa:Metadata</code> element
136     *
137     */
138     public W3CEndpointReferenceBuilder serviceName(QName serviceName) {
139         this.serviceName = serviceName;
140         return this;
141     }
142
143     /**
144     * Sets the <code>endpointName</code> as
145     * <code>wsam:ServiceName/@EndpointName</code> in the
146     * <code>wsa:Metadata</code> element. This method can only be called
147     * after the {@link #serviceName} method has been called.
148     * <p>
149     * See <a href="http://www.w3.org/TR/2007/REC-ws-addr-metadata-20070904/#refmetadatfromepr">
150     * 2.1 Referencing WSDL Metadata from an EPR</a> for more details.
151     *
152     * @param endpointName The name of the endpoint to be targeted
153     * by the returned <code>W3CEndpointReference</code>. The
154     * <code>endpointName</code> (portName) property may also be
155     * used with the <code>serviceName</code> property to lookup
156     * the <code>address</code> of a web service
157     * endpoint published by the same Java EE application.
158     *
159     * @return A <code>W3CEndpointReferenceBuilder</code> instance with
160     * the <code>endpointName</code> as
161     * <code>wsam:ServiceName/@EndpointName</code> in the
162     * <code>wsa:Metadata</code> element.

```



```

163     *
164     * @throws IllegalStateException, if the <code>serviceName</code>
165     * has not been set.
166     * @throws IllegalArgumentException, if the <code>endpointName</code>'s
167     * Namespace URI doesn't match <code>serviceName</code>'s Namespace URI
168     *
169     */
170     public W3CEndpointReferenceBuilder endpointName(QName endpointName) {
171         if (serviceName == null) {
172             throw new IllegalStateException("The W3CEndpointReferenceBuilder's serviceName must be
173                 set before setting the endpointName: "+endpointName);
174         }
175         this.endpointName = endpointName;
176         return this;
177     }
178
179     /**
180     * Sets the <code>wsdlDocumentLocation</code> that will be referenced
181     * as <code>wsa:Metadata/@wsdl:wsdlLocation</code>. The namespace name
182     * for the wsdl:wsdlLocation's value can be taken from the WSDL itself.
183     *
184     * <p>
185     * See <a href="http://www.w3.org/TR/2007/REC-ws-addr-metadata-20070904/#refmetadatfromepr">
186     * 2.1 Referencing WSDL Metadata from an EPR</a> for more details.
187     *
188     * @param wsdlDocumentLocation The location of the WSDL document to
189     *     be referenced in the <code>wsa:Metadata</code> of the
190     *     <code>W3CEndpointReference</code>.
191     * @return A <code>W3CEndpointReferenceBuilder</code> instance with
192     *     the <code>wsdlDocumentLocation</code> that is to be referenced.
193     */
194     public W3CEndpointReferenceBuilder wsdlDocumentLocation(String wsdlDocumentLocation) {
195         this.wsdlDocumentLocation = wsdlDocumentLocation;
196         return this;
197     }
198
199     /**
200     * Adds the <code>referenceParameter</code> to the
201     * <code>W3CEndpointReference</code> instance
202     * <code>wsa:ReferenceParameters</code> element.
203     *
204     * @param referenceParameter The element to be added to the
205     *     <code>wsa:ReferenceParameters</code> element.
206     *
207     * @return A <code>W3CEndpointReferenceBuilder</code> instance with
208     *     the <code>referenceParameter</code> added to the
209     *     <code>wsa:ReferenceParameters</code> element.
210     *
211     * @throws java.lang.IllegalArgumentException if <code>referenceParameter</code>
212     * is <code>null</code>.
213     */
214     public W3CEndpointReferenceBuilder referenceParameter(Element referenceParameter) {

```

```

215         if (referenceParameter == null)
216             throw new java.lang.IllegalArgumentException("The referenceParameter cannot be null.");
217         referenceParameters.add(referenceParameter);
218         return this;
219     }
220
221     /**
222      * Adds the <code>metadataElement</code> to the
223      * <code>W3CEndpointReference</code> instance's
224      * <code>wsa:Metadata</code> element.
225      *
226      * @param metadataElement The element to be added to the
227      *     <code>wsa:Metadata</code> element.
228      *
229      * @return A <code>W3CEndpointReferenceBuilder</code> instance with
230      *     the <code>metadataElement</code> added to the
231      *     <code>wsa:Metadata</code> element.
232      *
233      * @throws java.lang.IllegalArgumentException if <code>metadataElement</code>
234      *     is <code>null</code>.
235      */
236     public W3CEndpointReferenceBuilder metadata(Element metadataElement) {
237         if (metadataElement == null)
238             throw new java.lang.IllegalArgumentException("The metadataElement cannot be null.");
239         metadata.add(metadataElement);
240         return this;
241     }
242
243     /**
244      * Adds an extension element to the
245      * <code>W3CEndpointReference</code> instance's
246      * <code>wsa:EndpointReference</code> element.
247      *
248      * @param element The extension element to be added to the
249      *     <code>W3CEndpointReference</code>
250      * @return A <code>W3CEndpointReferenceBuilder</code> instance with
251      *     the extension <code>element</code> added to the
252      *     <code>W3CEndpointReference</code> instance.
253      * @throws java.lang.IllegalArgumentException if <code>element</code>
254      *     is <code>null</code>.
255      *
256      * @since JAX-WS 2.2
257      */
258     public W3CEndpointReferenceBuilder element(Element element) {
259         if (element == null) {
260             throw new IllegalArgumentException("The extension element cannot be null.");
261         }
262         elements.add(element);
263         return this;
264     }
265
266     /**
267      * Adds an extension attribute to the

```

```

268 * <code>W3CEndpointReference</code> instance's
269 * <code>wsa:EndpointReference</code> element.
270 *
271 * @param name The name of the extension attribute to be added to the
272 * <code>W3CEndpointReference</code>
273 * @param value extension attribute value
274 * @return A <code>W3CEndpointReferenceBuilder</code> instance with
275 * the extension attribute added to the <code>W3CEndpointReference</code>
276 * instance.
277 * @throws java.lang.IllegalArgumentException if <code>name</code>
278 * or <code>value</code> is <code>null</code>.
279 *
280 * @since JAX-WS 2.2
281 */
282 public W3CEndpointReferenceBuilder attribute(QName name, String value) {
283     if (name == null || value == null) {
284         throw new IllegalArgumentException("The extension attribute name or value cannot be
285             null.");
286     }
287     attributes.put(name, value);
288     return this;
289 }
290 /**
291 * Builds a <code>W3CEndpointReference</code> from the accumulated
292 * properties set on this <code>W3CEndpointReferenceBuilder</code>
293 * instance.
294 * <p>
295 * This method can be used to create a <code>W3CEndpointReference</code>
296 * for any endpoint by specifying the <code>address</code> property along
297 * with any other desired properties. This method
298 * can also be used to create a <code>W3CEndpointReference</code> for
299 * an endpoint that is published by the same Java EE application.
300 * This method can automatically determine the <code>address</code> of
301 * an endpoint published by the same Java EE application that is identified by the
302 * <code>serviceName</code> and
303 * <code>endpointName</code> properties. If the <code>address</code> is
304 * <code>null</code> and the <code>serviceName</code> and
305 * <code>endpointName</code>
306 * do not identify an endpoint published by the same Java EE application, a
307 * <code>java.lang.IllegalStateException</code> MUST be thrown.
308 *
309 *
310 * @return <code>W3CEndpointReference</code> from the accumulated
311 * properties set on this <code>W3CEndpointReferenceBuilder</code>
312 * instance. This method never returns <code>null</code>.
313 *
314 * @throws IllegalStateException
315 * <ul>
316 * <li>If the <code>address</code>, <code>serviceName</code> and
317 * <code>endpointName</code> are all <code>null</code>.
318 * <li>If the <code>serviceName</code> service is <code>null</code> and the
319 * <code>endpointName</code> is NOT <code>null</code>.

```

```

320 *      <li>If the <code>address</code> property is <code>null</code> and
321 *      the <code>serviceName</code> and <code>endpointName</code> do not
322 *      specify a valid endpoint published by the same Java EE
323 *      application.
324 *      <li>If the <code>serviceName</code> is NOT <code>null</code>
325 *      and is not present in the specified WSDL.
326 *      <li>If the <code>endpointName</code> port is not <code>null</code> and it
327 *      is not present in <code>serviceName</code> service in the WSDL.
328 *      <li>If the <code>wsdlDocumentLocation</code> is NOT <code>null</code>
329 *      and does not represent a valid WSDL.
330 *    </ul>
331 * @throws WebServiceException If an error occurs while creating the
332 *      <code>W3CEndpointReference</code>.
333 *
334 */
335 public W3CEndpointReference build() {
336     if (elements.isEmpty() && attributes.isEmpty() && interfaceName == null) {
337         // 2.1 API
338         return Provider.provider().createW3CEndpointReference(address,
339             serviceName, endpointName, metadata, wsdlDocumentLocation,
340             referenceParameters);
341     }
342     return Provider.provider().createW3CEndpointReference(address,
343         interfaceName, serviceName, endpointName, metadata, wsdlDocumentLocation,
344         referenceParameters, elements, attributes);
345 }
346
347 private String address;
348 private List<Element> referenceParameters;
349 private List<Element> metadata;
350 private QName interfaceName;
351 private QName serviceName;
352 private QName endpointName;
353 private String wsdlDocumentLocation;
354 private Map<QName,String> attributes;
355 private List<Element> elements;
356 }

```

### 34.3 package-info.java

Secuencia 318: javax/xml/ws/wsaddressing/package-info.java

```

1  /*
2  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```

13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  @javax.xml.bind.annotation.XmlSchema(namespace=W3CEndpointReference.NS,
27                                     location="http://www.w3.org/2006/03/addressing/ws-addr.xsd")
28  package javax.xml.ws.wsaddressing;

```

## 35 Xml Xpath (javax.xml.xpath package)

### 35.1 SecuritySupport.java

Secuencia 319: javax.xml.xpath/SecuritySupport.java

```

1  /*
2  * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 import java.net.URL;
29 import java.security.*;
30 import java.net.*;
31 import java.io.*;

```

```
32 import java.util.*;
33
34 /**
35  * This class is duplicated for each JAXP subpackage so keep it in sync.
36  * It is package private and therefore is not exposed as part of the JAXP
37  * API.
38  *
39  * Security related methods that only work on J2SE 1.2 and newer.
40  */
41 class SecuritySupport {
42
43
44     ClassLoader getContextClassLoader() {
45         return (ClassLoader)
46             AccessController.doPrivileged(new PrivilegedAction() {
47                 public Object run() {
48                     ClassLoader cl = null;
49                     try {
50                         cl = Thread.currentThread().getContextClassLoader();
51                     } catch (SecurityException ex) { }
52                     return cl;
53                 }
54             });
55     }
56
57     String getSystemProperty(final String propName) {
58         return (String)
59             AccessController.doPrivileged(new PrivilegedAction() {
60                 public Object run() {
61                     return System.getProperty(propName);
62                 }
63             });
64     }
65
66     FileInputStream getFileInputStream(final File file)
67         throws FileNotFoundException
68     {
69         try {
70             return (FileInputStream)
71                 AccessController.doPrivileged(new PrivilegedExceptionAction() {
72                     public Object run() throws FileNotFoundException {
73                         return new FileInputStream(file);
74                     }
75                 });
76         } catch (PrivilegedActionException e) {
77             throw (FileNotFoundException)e.getException();
78         }
79     }
80
81     InputStream getURLInputStream(final URL url)
82         throws IOException
83     {
84         try {
```

```
85         return (InputStream)
86             AccessController.doPrivileged(new PrivilegedExceptionAction() {
87                 public Object run() throws IOException {
88                     return url.openStream();
89                 }
90             });
91     } catch (PrivilegedActionException e) {
92         throw (IOException)e.getException();
93     }
94 }
95
96 URL getResourceAsURL(final ClassLoader cl,
97                     final String name)
98 {
99     return (URL)
100         AccessController.doPrivileged(new PrivilegedAction() {
101             public Object run() {
102                 URL url;
103                 if (cl == null) {
104                     url = Object.class.getResource(name);
105                 } else {
106                     url = cl.getResource(name);
107                 }
108                 return url;
109             }
110         });
111 }
112
113 Enumeration getResources(final ClassLoader cl,
114                          final String name) throws IOException
115 {
116     try{
117         return (Enumeration)
118             AccessController.doPrivileged(new PrivilegedExceptionAction() {
119                 public Object run() throws IOException{
120                     Enumeration enumeration;
121                     if (cl == null) {
122                         enumeration = ClassLoader.getSystemResources(name);
123                     } else {
124                         enumeration = cl.getResources(name);
125                     }
126                     return enumeration;
127                 }
128             });
129     }catch(PrivilegedActionException e){
130         throw (IOException)e.getException();
131     }
132 }
133
134 InputStream getResourceAsStream(final ClassLoader cl,
135                                 final String name)
136 {
137     return (InputStream)
```

```

138     AccessController.doPrivileged(new PrivilegedAction() {
139         public Object run() {
140             InputStream ris;
141             if (cl == null) {
142                 ris = Object.class.getResourceAsStream(name);
143             } else {
144                 ris = cl.getResourceAsStream(name);
145             }
146             return ris;
147         }
148     });
149 }
150
151 boolean doesFileExist(final File f) {
152     return ((Boolean)
153         AccessController.doPrivileged(new PrivilegedAction() {
154             public Object run() {
155                 return new Boolean(f.exists());
156             }
157         })).booleanValue();
158 }
159
160 }

```

## 35.2 XPath.java

Secuencia 320: javax.xml.xpath/XPath.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;

```



```

27
28 import org.xml.sax.InputSource;
29 import javax.xml.namespace.QName;
30 import javax.xml.namespace.NamespaceContext;
31
32 /**
33  * <p><code>XPath</code> provides access to the XPath evaluation environment and expressions.</p>
34  *
35  * <a name="XPath-evaluation"/>
36  * <table border="1" cellpadding="2">
37  *   <thead>
38  *     <tr>
39  *       <th colspan="2">Evaluation of XPath Expressions.</th>
40  *     </tr>
41  *   </thead>
42  *   <tbody>
43  *     <tr>
44  *       <td>context</td>
45  *       <td>
46  *         If a request is made to evaluate the expression in the absence
47  *         of a context item, an empty document node will be used for the context.
48  *         For the purposes of evaluating XPath expressions, a DocumentFragment
49  *         is treated like a Document node.
50  *       </td>
51  *     </tr>
52  *     <tr>
53  *       <td>variables</td>
54  *       <td>
55  *         If the expression contains a variable reference, its value will be found through the {
56  *         @link XPathVariableResolver}
57  *         set with {@link #setXPathVariableResolver(XPathVariableResolver resolver)}.
58  *         An {@link XPathExpressionException} is raised if the variable resolver is undefined or
59  *         the resolver returns <code>null</code> for the variable.
60  *         The value of a variable must be immutable through the course of any single evaluation.</p>
61  *       </td>
62  *     </tr>
63  *     <tr>
64  *       <td>functions</td>
65  *       <td>
66  *         If the expression contains a function reference, the function will be found through the {
67  *         @link XPathFunctionResolver}
68  *         set with {@link #setXPathFunctionResolver(XPathFunctionResolver resolver)}.
69  *         An {@link XPathExpressionException} is raised if the function resolver is undefined or
70  *         the function resolver returns <code>null</code> for the function.</p>
71  *       </td>
72  *     </tr>
73  *     <tr>
74  *       <td>QNames</td>
75  *       <td>
76  *         QNames in the expression are resolved against the XPath namespace context
77  *         set with {@link #setNamespaceContext(NamespaceContext nsContext)}.
78  *       </td>
79  *     </tr>
80  *   </tbody>
81  * </table>
82  */

```

```

77 * </tr>
78 * <tr>
79 * <td>result</td>
80 * <td>
81 * This result of evaluating an expression is converted to an instance of the desired return
   type.
82 * Valid return types are defined in {@link XPathConstants}.
83 * Conversion to the return type follows XPath conversion rules.</p>
84 * </td>
85 * </tr>
86 * </table>
87 *
88 * <p>An XPath object is not thread-safe and not reentrant.
89 * In other words, it is the application's responsibility to make
90 * sure that one {@link XPath} object is not used from
91 * more than one thread at any given time, and while the <code>evaluate</code>
92 * method is invoked, applications may not recursively call
93 * the <code>evaluate</code> method.
94 * <p>
95 *
96 * @author <a href="Norman.Walsh@Sun.com">Norman Walsh</a>
97 * @author <a href="Jeff.Suttor@Sun.com">Jeff Suttor</a>
98 * @see <a href="http://www.w3.org/TR/xpath">XML Path Language (XPath) Version 1.0</a>
99 * @since 1.5
100 */
101 public interface XPath {
102
103     /**
104      * <p>Reset this <code>XPath</code> to its original configuration.</p>
105      *
106      * <p><code>XPath</code> is reset to the same state as when it was created with
107      * {@link XPathFactory#newXPath()}.
108      * <code>reset()</code> is designed to allow the reuse of existing <code>XPath</code>s
109      * thus saving resources associated with the creation of new <code>XPath</code>s.</p>
110      *
111      * <p>The reset <code>XPath</code> is not guaranteed to have the same {@link
112      * XPathFunctionResolver}, {@link XPathVariableResolver}
113      * or {@link NamespaceContext} <code>Object</code>s, e.g. {@link Object#equals(Object obj)
114      * }.
115      * It is guaranteed to have a functionally equal <code>XPathFunctionResolver</code>, <code>
116      * XPathVariableResolver</code>
117      * and <code>NamespaceContext</code>.</p>
118      */
119     public void reset();
120
121     /**
122      * <p>Establish a variable resolver.</p>
123      *
124      * <p>A <code>NullPointerException</code> is thrown if <code>resolver</code> is <code>null</code>
125      * </p>
126      *
127      * @param resolver Variable resolver.
128      */

```

```
125     * @throws NullPointerException If <code>resolver</code> is <code>>null</code>.
126     */
127     public void setXPathVariableResolver(XPathVariableResolver resolver);
128
129     /**
130      * <p>Return the current variable resolver.</p>
131      *
132      * <p><code>>null</code> is returned in no variable resolver is in effect.</p>
133      *
134      * @return Current variable resolver.
135      */
136     public XPathVariableResolver getXPathVariableResolver();
137
138     /**
139      * <p>Establish a function resolver.</p>
140      *
141      * <p>A <code>NullPointerException</code> is thrown if <code>resolver</code> is <code>>null</code>
142      * <code></p>
143      * @param resolver XPath function resolver.
144      *
145      * @throws NullPointerException If <code>resolver</code> is <code>>null</code>.
146      */
147     public void setXPathFunctionResolver(XPathFunctionResolver resolver);
148
149     /**
150      * <p>Return the current function resolver.</p>
151      *
152      * <p><code>>null</code> is returned in no function resolver is in effect.</p>
153      *
154      * @return Current function resolver.
155      */
156     public XPathFunctionResolver getXPathFunctionResolver();
157
158     /**
159      * <p>Establish a namespace context.</p>
160      *
161      * <p>A <code>NullPointerException</code> is thrown if <code>nsContext</code> is <code>>null</code>
162      * <code></p>
163      * @param nsContext Namespace context to use.
164      *
165      * @throws NullPointerException If <code>nsContext</code> is <code>>null</code>.
166      */
167     public void setNamespaceContext(NamespaceContext nsContext);
168
169     /**
170      * <p>Return the current namespace context.</p>
171      *
172      * <p><code>>null</code> is returned in no namespace context is in effect.</p>
173      *
174      * @return Current Namespace context.
175      */
```

```

176 public NamespaceContext getNamespaceContext();
177
178 /**
179  * <p>Compile an XPath expression for later evaluation.</p>
180  *
181  * <p>If <code>expression</code> contains any {@link XPathFunction}s,
182  * they must be available via the {@link XPathFunctionResolver}.
183  * An {@link XPathExpressionException} will be thrown if the
184  * <code>XPathFunction</code>
185  * cannot be resolved with the <code>XPathFunctionResolver</code>.</p>
186  *
187  * <p>If <code>expression</code> contains any variables, the
188  * {@link XPathVariableResolver} in effect
189  * <strong>at compile time</strong> will be used to resolve them.</p>
190  *
191  * <p>If <code>expression</code> is <code>null</code>, a <code>NullPointerException</code> is
192  *   thrown.</p>
193  *
194  * @param expression The XPath expression.
195  *
196  * @return Compiled XPath expression.
197  *
198  * @throws XPathExpressionException If <code>expression</code> cannot be compiled.
199  * @throws NullPointerException If <code>expression</code> is <code>null</code>.
200  */
201 public XPathExpression compile(String expression)
202     throws XPathExpressionException;
203
204 /**
205  * <p>Evaluate an <code>XPath</code> expression in the specified context and return the result
206  *   as the specified type.</p>
207  *
208  * <p>See <a href="#XPath-evaluation">Evaluation of XPath Expressions</a> for context item
209  *   evaluation,
210  * variable, function and <code>QName</code> resolution and return type conversion.</p>
211  *
212  * <p>If <code>returnType</code> is not one of the types defined in {@link XPathConstants} (
213  *   {@link XPathConstants#NUMBER NUMBER},
214  *   {@link XPathConstants#STRING STRING},
215  *   {@link XPathConstants#BOOLEAN BOOLEAN},
216  *   {@link XPathConstants#NODE NODE} or
217  *   {@link XPathConstants#NODESET NODESET})
218  * then an <code>IllegalArgumentException</code> is thrown.</p>
219  *
220  * <p>If a <code>null</code> value is provided for
221  * <code>item</code>, an empty document will be used for the
222  * context.
223  * If <code>expression</code> or <code>returnType</code> is <code>null</code>, then a
224  * <code>NullPointerException</code> is thrown.</p>
225  *
226  * @param expression The XPath expression.
227  * @param item The starting context (a node, for example).
228  * @param returnType The desired return type.

```

```

226 *
227 * @return Result of evaluating an XPath expression as an Object of returnType.
228 *
229 * @throws XPathExpressionException If expression cannot be evaluated.
230 * @throws IllegalArgumentException If returnType is not one of the types defined
231 *   in {@link XPathConstants}.
232 * @throws NullPointerException If expression or returnType is null.
233 */
234 public Object evaluate(String expression, Object item, QName returnType)
235     throws XPathExpressionException;
236
237 /**
238 * <p>Evaluate an XPath expression in the specified context and return the result as a String</p>
239 *
240 * <p>This method calls {@link #evaluate(String expression, Object item, QName returnType)}
241 *   with a returnType of
242 *   {@link XPathConstants#STRING}</p>
243 *
244 * <p>See <a href="#XPath-evaluation">Evaluation of XPath Expressions</a> for context item
245 *   evaluation,
246 *   variable, function and QName resolution and return type conversion.</p>
247 *
248 * <p>If a null value is provided for
249 *   item, an empty document will be used for the
250 *   context.
251 *   If expression is null, then a NullPointerException is
252 *   thrown.</p>
253 *
254 * @param expression The XPath expression.
255 * @param item The starting context (a node, for example).
256 *
257 * @return The String that is the result of evaluating the expression and
258 *   converting the result to a String.
259 *
260 * @throws XPathExpressionException If expression cannot be evaluated.
261 * @throws NullPointerException If expression is null.
262 */
263 public String evaluate(String expression, Object item)
264     throws XPathExpressionException;
265
266 /**
267 * <p>Evaluate an XPath expression in the context of the specified InputSource
268 *   and return the result as the specified type.</p>
269 *
270 * <p>This method builds a data model for the {@link InputSource} and calls
271 *   {@link #evaluate(String expression, Object item, QName returnType)} on the resulting
272 *   document object.</p>
273 *
274 * <p>See <a href="#XPath-evaluation">Evaluation of XPath Expressions</a> for context item
275 *   evaluation,

```

```

270     * variable, function and QName resolution and return type conversion.</p>
271     *
272     * <p>If <code>returnType</code> is not one of the types defined in {@link XPathConstants},
273     * then an <code>IllegalArgumentException</code> is thrown.</p>
274     *
275     * <p>If <code>expression</code>, <code>source</code> or <code>returnType</code> is <code>null
276     * </code>,
277     * then a <code>NullPointerException</code> is thrown.</p>
278     *
279     * @param expression The XPath expression.
280     * @param source The input source of the document to evaluate over.
281     * @param returnType The desired return type.
282     * @return The <code>Object</code> that encapsulates the result of evaluating the expression.
283     *
284     * @throws XPathExpressionException If expression cannot be evaluated.
285     * @throws IllegalArgumentException If <code>returnType</code> is not one of the types defined
286     *   in {@link XPathConstants}.
287     * @throws NullPointerException If <code>expression</code>, <code>source</code> or <code>
288     *   returnType</code>
289     *   is <code>null</code>.
290     */
291     public Object evaluate(
292         String expression,
293         InputSource source,
294         QName returnType)
295         throws XPathExpressionException;
296
297     /**
298     * <p>Evaluate an XPath expression in the context of the specified <code>InputSource</code>
299     * and return the result as a <code>String</code>.</p>
300     *
301     * <p>This method calls {@link #evaluate(String expression, InputSource source, QName
302     *   returnType)} with a
303     * <code>returnType</code> of {@link XPathConstants#STRING}.</p>
304     *
305     * <p>See <a href="#XPath-evaluation">Evaluation of XPath Expressions</a> for context item
306     * evaluation,
307     * variable, function and QName resolution and return type conversion.</p>
308     *
309     * <p>If <code>expression</code> or <code>source</code> is <code>null</code>,
310     * then a <code>NullPointerException</code> is thrown.</p>
311     *
312     * @param expression The XPath expression.
313     * @param source The <code>InputSource</code> of the document to evaluate over.
314     * @return The <code>String</code> that is the result of evaluating the expression and
315     *   converting the result to a <code>String</code>.
316     *
317     * @throws XPathExpressionException If expression cannot be evaluated.
318     * @throws NullPointerException If <code>expression</code> or <code>source</code> is <code>null
319     *   </code>.
320     */

```

```

317     public String evaluate(String expression, InputSource source)
318         throws XPathExpressionException;
319 }

```

### 35.3 XPathConstants.java

Secuencia 321: javax/xml/xpath/XPathConstants.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 import javax.xml.namespace.QName;
29
30 /**
31  * <p>XPath constants.</p>
32  *
33  * @author <a href="mailto:Norman.Walsh@Sun.COM">Norman Walsh</a>
34  * @author <a href="mailto:Jeff.Suttor@Sun.COM">Jeff Suttor</a>
35  * @see <a href="http://www.w3.org/TR/xpath">XML Path Language (XPath) Version 1.0</a>
36  * @since 1.5
37  */
38 public class XPathConstants {
39
40     /**
41      * <p>Private constructor to prevent instantiation.</p>
42      */
43     private XPathConstants() { }
44
45     /**
46      * <p>The XPath 1.0 number data type.</p>

```

```

47      *
48      * <p>Maps to Java {@link Double}.</p>
49      */
50      public static final QName NUMBER = new QName("http://www.w3.org/1999/XSL/Transform", "NUMBER");
51
52      /**
53      * <p>The XPath 1.0 string data type.</p>
54      *
55      * <p>Maps to Java {@link String}.</p>
56      */
57      public static final QName STRING = new QName("http://www.w3.org/1999/XSL/Transform", "STRING");
58
59      /**
60      * <p>The XPath 1.0 boolean data type.</p>
61      *
62      * <p>Maps to Java {@link Boolean}.</p>
63      */
64      public static final QName BOOLEAN = new QName("http://www.w3.org/1999/XSL/Transform", "BOOLEAN");
65
66      /**
67      * <p>The XPath 1.0 NodeSet data type.</p>
68      *
69      * <p>Maps to Java {@link org.w3c.dom.NodeList}.</p>
70      */
71      public static final QName NODESET = new QName("http://www.w3.org/1999/XSL/Transform", "NODESET");
72
73      /**
74      * <p>The XPath 1.0 NodeSet data type.
75      *
76      * <p>Maps to Java {@link org.w3c.dom.Node}.</p>
77      */
78      public static final QName NODE = new QName("http://www.w3.org/1999/XSL/Transform", "NODE");
79
80      /**
81      * <p>The URI for the DOM object model, "http://java.sun.com/jaxp/xpath/dom".</p>
82      */
83      public static final String DOM_OBJECT_MODEL = "http://java.sun.com/jaxp/xpath/dom";
84  }

```

### 35.4 XPathException.java

Secuencia 322: javax/xml/xpath/XPathException.java

```

1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *

```



```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  package javax.xml.xpath;
27
28  import java.io.PrintWriter;
29  import java.io.IOException;
30  import java.io.ObjectInputStream;
31  import java.io.ObjectOutputStream;
32  import java.io.ObjectStreamField;
33  import java.io.InvalidClassException;
34
35  /**
36   * <code>XPathException</code> represents a generic XPath exception.</p>
37   *
38   * @author <a href="Norman.Walsh@Sun.com">Norman Walsh</a>
39   * @author <a href="mailto:Jeff.Suttor@Sun.COM">Jeff Suttor</a>
40   * @since 1.5
41   */
42  public class XPathException extends Exception {
43
44      private static final ObjectStreamField[] serialPersistentFields = {
45          new ObjectStreamField( "cause", Throwable.class )
46      };
47
48      /**
49       * <p>Stream Unique Identifier.</p>
50       */
51      private static final long serialVersionUID = -1837080260374986980L;
52
53      /**
54       * <p>Constructs a new <code>XPathException</code>
55       * with the specified detail <code>message</code>.</p>
56       *
57       * <p>The <code>cause</code> is not initialized.</p>
58       *
59       * <p>If <code>message</code> is <code>null</code>,
60       * then a <code>NullPointerException</code> is thrown.</p>
61       *
62       * @param message The detail message.

```

```

63      *
64      * @throws NullPointerException When <code>message</code> is
65      *   <code>null</code>.
66      */
67      public XPathException(String message) {
68          super(message);
69          if ( message == null ) {
70              throw new NullPointerException ( "message can't be null");
71          }
72      }
73
74      /**
75       * <p>Constructs a new <code>XPathException</code>
76       * with the specified <code>cause</code>.</p>
77       *
78       * <p>If <code>cause</code> is <code>null</code>,
79       * then a <code>NullPointerException</code> is thrown.</p>
80       *
81       * @param cause The cause.
82       *
83       * @throws NullPointerException if <code>cause</code> is <code>null</code>.
84       */
85      public XPathException(Throwable cause) {
86          super(cause);
87          if ( cause == null ) {
88              throw new NullPointerException ( "cause can't be null");
89          }
90      }
91
92      /**
93       * <p>Get the cause of this XPathException.</p>
94       *
95       * @return Cause of this XPathException.
96       */
97      public Throwable getCause() {
98          return super.getCause();
99      }
100
101      /**
102       * Writes "cause" field to the stream.
103       * The cause is got from the parent class.
104       *
105       * @param out stream used for serialization.
106       * @throws IOException thrown by <code>ObjectOutputStream</code>
107       *
108       */
109      private void writeObject(ObjectOutputStream out)
110          throws IOException
111      {
112          ObjectOutputStream.PutField fields = out.putFields();
113          fields.put("cause", (Throwable) super.getCause());
114          out.writeFields();
115      }

```

```

116
117 /**
118  * Reads the "cause" field from the stream.
119  * And initializes the "cause" if it wasn't
120  * done before.
121  *
122  * @param in stream used for deserialization
123  * @throws IOException thrown by <code>ObjectInputStream</code>
124  * @throws ClassNotFoundException thrown by <code>ObjectInputStream</code>
125  */
126 private void readObject(ObjectInputStream in)
127     throws IOException, ClassNotFoundException
128 {
129     ObjectInputStream.GetField fields = in.readFields();
130     Throwable scause = (Throwable) fields.get("cause", null);
131
132     if (super.getCause() == null && scause != null) {
133         try {
134             super.initCause(scause);
135         } catch (IllegalStateException e) {
136             throw new InvalidClassException("Inconsistent state: two causes");
137         }
138     }
139 }
140
141 /**
142  * <p>Print stack trace to specified <code>PrintStream</code>.</p>
143  *
144  * @param s Print stack trace to this <code>PrintStream</code>.
145  */
146 public void printStackTrace(java.io.PrintStream s) {
147     if (getCause() != null) {
148         getCause().printStackTrace(s);
149         s.println("----- linked to -----");
150     }
151
152     super.printStackTrace(s);
153 }
154
155 /**
156  * <p>Print stack trace to <code>System.err</code>.</p>
157  */
158 public void printStackTrace() {
159     printStackTrace(System.err);
160 }
161
162 /**
163  * <p>Print stack trace to specified <code>PrintWriter</code>.</p>
164  *
165  * @param s Print stack trace to this <code>PrintWriter</code>.
166  */
167 public void printStackTrace(PrintWriter s) {
168

```

```

169     if (getCause() != null) {
170         getCause().printStackTrace(s);
171         s.println("----- linked to -----");
172     }
173
174     super.printStackTrace(s);
175 }
176 }

```

### 35.5 XPathExpression.java

Secuencia 323: javax/xml/xpath/XPathExpression.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 import org.xml.sax.InputSource;
29 import javax.xml.namespace.QName;
30
31 /**
32 * <p><code>XPathExpression</code> provides access to compiled XPath expressions.</p>
33 *
34 * <a name="XPathExpression-evaluation">
35 * <table border="1" cellpadding="2">
36 *   <thead>
37 *     <tr>
38 *       <th colspan="2">Evaluation of XPath Expressions.</th>
39 *     </tr>
40 *   </thead>
41 *   <tbody>

```

```

42 *      <tr>
43 *          <td>context</td>
44 *          <td>
45 *              If a request is made to evaluate the expression in the absence
46 * of a context item, an empty document node will be used for the context.
47 * For the purposes of evaluating XPath expressions, a DocumentFragment
48 * is treated like a Document node.
49 *          </td>
50 *      </tr>
51 *      <tr>
52 *          <td>variables</td>
53 *          <td>
54 *              If the expression contains a variable reference, its value will be found through the {
55 * @link XPathVariableResolver}.
56 *              An {@link XPathExpressionException} is raised if the variable resolver is undefined or
57 * the resolver returns <code>null</code> for the variable.
58 *              The value of a variable must be immutable through the course of any single evaluation.</p>
59 *          </td>
60 *      </tr>
61 *      <tr>
62 *          <td>functions</td>
63 *          <td>
64 *              If the expression contains a function reference, the function will be found through the {
65 * @link XPathFunctionResolver}.
66 *              An {@link XPathExpressionException} is raised if the function resolver is undefined or
67 * the function resolver returns <code>null</code> for the function.</p>
68 *          </td>
69 *      </tr>
70 *      <tr>
71 *          <td>QNames</td>
72 *          <td>
73 *              QNames in the expression are resolved against the XPath namespace context.
74 *          </td>
75 *      </tr>
76 *      <tr>
77 *          <td>result</td>
78 *          <td>
79 *              This result of evaluating an expression is converted to an instance of the desired return
80 * type.
81 *              Valid return types are defined in {@link XPathConstants}.
82 *              Conversion to the return type follows XPath conversion rules.</p>
83 *          </td>
84 *      </tr>
85 * </table>
86 *
87 * <p>An XPath expression is not thread-safe and not reentrant.
88 * In other words, it is the application's responsibility to make
89 * sure that one {@link XPathExpression} object is not used from
90 * more than one thread at any given time, and while the <code>evaluate</code>
91 * method is invoked, applications may not recursively call
92 * the <code>evaluate</code> method.
93 * <p>

```

```

91  *
92  * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
93  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
94  * @see <a href="http://www.w3.org/TR/xpath#section-Expressions">XML Path Language (XPath) Version
    1.0, Expressions</a>
95  * @since 1.5
96  */
97  public interface XPathExpression {
98
99      /**
100       * <p>Evaluate the compiled XPath expression in the specified context and return the result as
        the specified type.</p>
101       *
102       * <p>See <a href="#XPathExpression-evaluation">Evaluation of XPath Expressions</a> for context
        item evaluation,
103       * variable, function and QName resolution and return type conversion.</p>
104       *
105       * <p>If <code>returnType</code> is not one of the types defined in {@link XPathConstants},
106       * then an <code>IllegalArgumentException</code> is thrown.</p>
107       *
108       * <p>If a <code>null</code> value is provided for
109       * <code>item</code>, an empty document will be used for the
110       * context.
111       * If <code>returnType</code> is <code>null</code>, then a <code>NullPointerException</code> is
        thrown.</p>
112       *
113       * @param item The starting context (a node, for example).
114       * @param returnType The desired return type.
115       *
116       * @return The <code>Object</code> that is the result of evaluating the expression and
        converting the result to
117       * <code>returnType</code>.
118       *
119       * @throws XPathExpressionException If the expression cannot be evaluated.
120       * @throws IllegalArgumentException If <code>returnType</code> is not one of the types defined
        in {@link XPathConstants}.
121       * @throws NullPointerException If <code>returnType</code> is <code>null</code>.
122       */
123     public Object evaluate(Object item, QName returnType)
124         throws XPathExpressionException;
125
126     /**
127       * <p>Evaluate the compiled XPath expression in the specified context and return the result as
        a <code>String</code>.</p>
128       *
129       * <p>This method calls {@link #evaluate(Object item, QName returnType)} with a <code>
        returnType</code> of
130       * {@link XPathConstants#STRING}.</p>
131       *
132       * <p>See <a href="#XPathExpression-evaluation">Evaluation of XPath Expressions</a> for context
        item evaluation,
133       * variable, function and QName resolution and return type conversion.</p>
134       *

```

```

135 * <p>If a <code>null</code> value is provided for
136 * <code>item</code>, an empty document will be used for the
137 * context.
138 *
139 * @param item The starting context (a node, for example).
140 *
141 * @return The <code>String</code> that is the result of evaluating the expression and
        converting the result to a
142 * <code>String</code>.
143 *
144 * @throws XPathExpressionException If the expression cannot be evaluated.
145 */
146 public String evaluate(Object item)
147     throws XPathExpressionException;
148
149 /**
150 * <p>Evaluate the compiled XPath expression in the context of the specified <code>InputSource
        </code> and return the result as the
151 * specified type.</p>
152 *
153 * <p>This method builds a data model for the {@link InputSource} and calls
154 * {@link #evaluate(Object item, QName returnType)} on the resulting document object.</p>
155 *
156 * <p>See <a href="#XPathExpression-evaluation">Evaluation of XPath Expressions</a> for context
        item evaluation,
157 * variable, function and QName resolution and return type conversion.</p>
158 *
159 * <p>If <code>returnType</code> is not one of the types defined in {@link XPathConstants},
160 * then an <code>IllegalArgumentException</code> is thrown.</p>
161 *
162 * <p>If <code>source</code> or <code>returnType</code> is <code>null</code>,
163 * then a <code>NullPointerException</code> is thrown.</p>
164 *
165 * @param source The <code>InputSource</code> of the document to evaluate over.
166 * @param returnType The desired return type.
167 *
168 * @return The <code>Object</code> that is the result of evaluating the expression and
        converting the result to
169 * <code>returnType</code>.
170 *
171 * @throws XPathExpressionException If the expression cannot be evaluated.
172 * @throws IllegalArgumentException If <code>returnType</code> is not one of the types defined
        in {@link XPathConstants}.
173 * @throws NullPointerException If <code>source</code> or <code>returnType</code> is <code>
        null</code>.
174 */
175 public Object evaluate(InputSource source, QName returnType)
176     throws XPathExpressionException;
177
178 /**
179 * <p>Evaluate the compiled XPath expression in the context of the specified <code>InputSource
        </code> and return the result as a
180 * <code>String</code>.</p>

```

```

181      *
182      * <p>This method calls {@link #evaluate(InputSource source, QName returnType)} with a <code>
      returnType</code> of
183      * {@link XPathConstants#STRING}.</p>
184      *
185      * <p>See <a href="#XPathExpression-evaluation">Evaluation of XPath Expressions</a> for context
      item evaluation,
186      * variable, function and QName resolution and return type conversion.</p>
187      *
188      * <p>If <code>source</code> is <code>null</code>, then a <code>NullPointerException</code> is
      thrown.</p>
189      *
190      * @param source The <code>InputSource</code> of the document to evaluate over.
191      *
192      * @return The <code>String</code> that is the result of evaluating the expression and
      converting the result to a
193      * <code>String</code>.
194      *
195      * @throws XPathExpressionException If the expression cannot be evaluated.
196      * @throws NullPointerException If <code>source</code> is <code>null</code>.
197      */
198      public String evaluate(InputSource source)
199          throws XPathExpressionException;
200  }

```

### 35.6 XPathExpressionException.java

Secuencia 324: javax/xml/xpath/XPathExpressionException.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25

```



```

26 package javax.xml.xpath;
27
28 /**
29  * <code>XPathExpressionException</code> represents an error in an XPath expression.</p>
30  *
31  * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
32  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
33  * @since 1.5
34  */
35 public class XPathExpressionException extends XPathException {
36
37     /**
38      * <p>Stream Unique Identifier.</p>
39      */
40     private static final long serialVersionUID = -1837080260374986980L;
41
42     /**
43      * <p>Constructs a new <code>XPathExpressionException</code>
44      * with the specified detail <code>message</code>.</p>
45      *
46      * <p>The <code>cause</code> is not initialized.</p>
47      *
48      * <p>If <code>message</code> is <code>null</code>,
49      * then a <code>NullPointerException</code> is thrown.</p>
50      *
51      * @param message The detail message.
52      *
53      * @throws NullPointerException When <code>message</code> is
54      * <code>null</code>.
55      */
56     public XPathExpressionException(String message) {
57         super(message);
58     }
59
60     /**
61      * <p>Constructs a new <code>XPathExpressionException</code>
62      * with the specified <code>cause</code>.</p>
63      *
64      * <p>If <code>cause</code> is <code>null</code>,
65      * then a <code>NullPointerException</code> is thrown.</p>
66      *
67      * @param cause The cause.
68      *
69      * @throws NullPointerException if <code>cause</code> is <code>null</code>.
70      */
71     public XPathExpressionException(Throwable cause) {
72         super(cause);
73     }
74 }

```

## 35.7 XPathFactory.java

Secuencia 325: javax/xml/xpath/XPathFactory.java

```
1  /*
2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 /**
29  * <p>An XPathFactory instance can be used to create
30  * {@link javax.xml.xpath.XPath} objects.</p>
31  *
32  * <p>See {@link #newInstance(String uri)} for lookup mechanism.</p>
33  *
34  * <p>The {@link XPathFactory} class is not thread-safe. In other words,
35  * it is the application's responsibility to ensure that at most
36  * one thread is using a {@link XPathFactory} object at any
37  * given moment. Implementations are encouraged to mark methods
38  * as synchronized to protect themselves from broken clients.
39  *
40  * <p>{@link XPathFactory} is not re-entrant. While one of the
41  * newInstance methods is being invoked, applications
42  * may not attempt to recursively invoke a newInstance method,
43  * even from the same thread.
44  *
45  * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
46  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
47  *
48  * @since 1.5
49  */
50 public abstract class XPathFactory {
51
52
53     /**
```

```

54     * <p>The default property name according to the JAXP spec.</p>
55     */
56     public static final String DEFAULT_PROPERTY_NAME = "javax.xml.xpath.XPathFactory";
57
58     /**
59     * <p>Default Object Model URI.</p>
60     */
61     public static final String DEFAULT_OBJECT_MODEL_URI = "http://java.sun.com/jaxp/xpath/dom";
62
63     /**
64     * <p> Take care of restrictions imposed by java security model </p>
65     */
66     private static SecuritySupport ss = new SecuritySupport() ;
67
68     /**
69     * <p>Protected constructor as {@link #newInstance()} or {@link #newInstance(String uri)}
70     * or {@link #newInstance(String uri, String factoryClassName, ClassLoader classLoader)}
71     * should be used to create a new instance of an <code>XPathFactory</code>.</p>
72     */
73     protected XPathFactory() {
74     }
75
76     /**
77     * <p>Get a new <code>XPathFactory</code> instance using the default object model,
78     * {@link #DEFAULT_OBJECT_MODEL_URI},
79     * the W3C DOM.</p>
80     *
81     * <p>This method is functionally equivalent to:</p>
82     * <pre>
83     *     newInstance(DEFAULT_OBJECT_MODEL_URI)
84     * </pre>
85     *
86     * <p>Since the implementation for the W3C DOM is always available, this method will never fail
87     * .</p>
88     *
89     * @return Instance of an <code>XPathFactory</code>.
90     *
91     * @throws RuntimeException When there is a failure in creating an
92     * <code>XPathFactory</code> for the default object model.
93     */
94     public static XPathFactory newInstance() {
95
96         try {
97             return newInstance(DEFAULT_OBJECT_MODEL_URI);
98         } catch (XPathFactoryConfigurationException xpathFactoryConfigurationException) {
99             throw new RuntimeException(
100                 "XPathFactory#newInstance() failed to create an XPathFactory for the
101                 default object model: "
102                 + DEFAULT_OBJECT_MODEL_URI
103                 + " with the XPathFactoryConfigurationException: "
104                 + xpathFactoryConfigurationException.toString()
105             );
106         }
107     }

```

```

105     }
106
107     /**
108     * <p>Get a new <code>XPathFactory</code> instance using the specified object model.</p>
109     *
110     * <p>To find a <code>XPathFactory</code> object,
111     * this method looks the following places in the following order where "the class loader" refers
112     * to the context class loader:</p>
113     * <ol>
114     *   <li>
115     *     If the system property {@link #DEFAULT_PROPERTY_NAME} + ":uri" is present,
116     *     where uri is the parameter to this method, then its value is read as a class name.
117     *     The method will try to create a new instance of this class by using the class loader,
118     *     and returns it if it is successfully created.
119     *   </li>
120     *   <li>
121     *     ${java.home}/lib/jaxp.properties is read and the value associated with the key being the
122     *     system property above is looked for.
123     *     If present, the value is processed just like above.
124     *   </li>
125     *   <li>
126     *     Use the service-provider loading facilities, defined by the
127     *     {@link java.util.ServiceLoader} class, to attempt to locate and load an
128     *     implementation of the service using the {@linkplain
129     *     java.util.ServiceLoader#load(java.lang.Class) default loading mechanism}:
130     *     the service-provider loading facility will use the {@linkplain
131     *     java.lang.Thread#getContextClassLoader() current thread's context class loader}
132     *     to attempt to load the service. If the context class
133     *     loader is null, the {@linkplain
134     *     ClassLoader#getSystemClassLoader() system class loader} will be used.
135     *   <br>
136     *     Each potential service provider is required to implement the method
137     *     {@link #isObjectModelSupported(String objectModel)}.
138     *     The first service provider found that supports the specified object
139     *     model is returned.
140     *   <br>
141     *     In case of {@link java.util.ServiceConfigurationError} an
142     *     {@link XPathFactoryConfigurationException} will be thrown.
143     *   </li>
144     *   <li>
145     *     Platform default <code>XPathFactory</code> is located in a platform specific way.
146     *     There must be a platform default XPathFactory for the W3C DOM, i.e. {@link #
147     *     DEFAULT_OBJECT_MODEL_URI}.
148     *   </li>
149     * </ol>
150     * <p>If everything fails, an <code>XPathFactoryConfigurationException</code> will be thrown.</p>
151     * <p>Tip for Trouble-shooting:</p>
152     * <p>See {@link java.util.Properties#load(java.io.InputStream)} for exactly how a property file
153     * is parsed.
154     *
155     * In particular, colons ':' need to be escaped in a property file, so make sure the URIs are
156     properly escaped in it.

```

```

152  * For example:</p>
153  * <pre>
154  *   http://java.sun.com/jaxp/xpath/dom=org.acme.DomXPathFactory
155  * </pre>
156  *
157  * @param uri Identifies the underlying object model.
158  *   The specification only defines the URI {@link #DEFAULT_OBJECT_MODEL_URI},
159  *   <code>http://java.sun.com/jaxp/xpath/dom</code> for the W3C DOM,
160  *   the org.w3c.dom package, and implementations are free to introduce other URIs for other
161  *   object models.
162  * @return Instance of an <code>XPathFactory</code>.
163  *
164  * @throws XPathFactoryConfigurationException If the specified object model
165  *   is unavailable, or if there is a configuration error.
166  * @throws NullPointerException If <code>uri</code> is <code>null</code>.
167  * @throws IllegalArgumentException If <code>uri</code> is <code>null</code>
168  *   or <code>uri.length() == 0</code>.
169  */
170  public static XPathFactory newInstance(final String uri)
171      throws XPathFactoryConfigurationException {
172
173      if (uri == null) {
174          throw new NullPointerException(
175              "XPathFactory#newInstance(String uri) cannot be called with uri == null");
176      }
177
178      if (uri.length() == 0) {
179          throw new IllegalArgumentException(
180              "XPathFactory#newInstance(String uri) cannot be called with uri == \"\");
181      }
182
183      ClassLoader classLoader = ss.getContextClassLoader();
184
185      if (classLoader == null) {
186          //use the current class loader
187          classLoader = XPathFactory.class.getClassLoader();
188      }
189
190      XPathFactory xpathFactory = new XPathFactoryFinder(classLoader).newFactory(uri);
191
192      if (xpathFactory == null) {
193          throw new XPathFactoryConfigurationException(
194              "No XPathFactory implementation found for the object model: "
195              + uri);
196      }
197
198      return xpathFactory;
199  }
200
201  /**
202  * <p>Obtain a new instance of a <code>XPathFactory</code> from a factory class name. <code>
203      XPathFactory</code>

```

```

203 * is returned if specified factory class supports the specified object model.
204 * This function is useful when there are multiple providers in the classpath.
205 * It gives more control to the application as it can specify which provider
206 * should be loaded.</p>
207 *
208 *
209 * <h2>Tip for Trouble-shooting</h2>
210 * <p>Setting the <code>jaxp.debug</code> system property will cause
211 * this method to print a lot of debug messages
212 * to <code>System.err</code> about what it is doing and where it is looking at.</p>
213 *
214 * <p> If you have problems try:</p>
215 * <pre>
216 * java -Djaxp.debug=1 YourProgram ....
217 * </pre>
218 *
219 * @param uri          Identifies the underlying object model. The specification only defines
220 *                      the URI
221 *                      {<code>@link #DEFAULT_OBJECT_MODEL_URI</code>, <code>http://java.sun.com/jaxp/xpath/
222 *                      dom</code>
223 *                      for the W3C DOM, the org.w3c.dom package, and implementations are free to
224 *                      introduce
225 *                      other URIs for other object models.
226 *
227 * @param factoryClassName fully qualified factory class name that provides implementation of <
228 *                      code>javax.xml.xpath.XPathFactory</code>.
229 *
230 * @param classLoader <code>ClassLoader</code> used to load the factory class. If <code>null</code>/
231 *                      code>
232 *                      current <code>Thread</code>'s context classLoader is used to load the
233 *                      factory class.
234 *
235 * @return New instance of a <code>XPathFactory</code>
236 *
237 * @throws XPathFactoryConfigurationException
238 *         if <code>factoryClassName</code> is <code>null</code>, or
239 *         the factory class cannot be loaded, instantiated
240 *         or the factory class does not support the object model specified
241 *         in the <code>uri</code> parameter.
242 *
243 * @throws NullPointerException If <code>uri</code> is <code>null</code>.
244 * @throws IllegalArgumentException If <code>uri</code> is <code>null</code>
245 *         or <code>uri.length() == 0</code>.
246 *
247 * @see #newInstance()
248 * @see #newInstance(String uri)
249 *
250 * @since 1.6
251 */
252 public static XPathFactory newInstance(String uri, String factoryClassName, ClassLoader
253                                     classLoader)
254     throws XPathFactoryConfigurationException{

```

```

249     ClassLoader cl = classLoader;
250
251     if (uri == null) {
252         throw new NullPointerException(
253             "XPathFactory#newInstance(String uri) cannot be called with uri == null");
254     }
255
256     if (uri.length() == 0) {
257         throw new IllegalArgumentException(
258             "XPathFactory#newInstance(String uri) cannot be called with uri == \"\"");
259     }
260
261     if (cl == null) {
262         cl = ss.getContextClassLoader();
263     }
264
265     XPathFactory f = new XPathFactoryFinder(cl).createInstance(factoryClassName);
266
267     if (f == null) {
268         throw new XPathFactoryConfigurationException(
269             "No XPathFactory implementation found for the object model: "
270             + uri);
271     }
272     //if this factory supports the given schemalanguage return this factory else thrown
273     //exception
274     if (f.isObjectModelSupported(uri)) {
275         return f;
276     } else {
277         throw new XPathFactoryConfigurationException("Factory "
278             + factoryClassName + " doesn't support given " + uri
279             + " object model");
280     }
281 }
282
283 /**
284  * <p>Is specified object model supported by this <code>XPathFactory</code>?</p>
285  *
286  * @param objectModel Specifies the object model which the returned <code>XPathFactory</code>
287  *    will understand.
288  *
289  * @return <code>true</code> if <code>XPathFactory</code> supports <code>objectModel</code>,
290  *    else <code>false</code>.
291  *
292  * @throws NullPointerException If <code>objectModel</code> is <code>null</code>.
293  * @throws IllegalArgumentException If <code>objectModel.length() == 0</code>.
294  */
295 public abstract boolean isObjectModelSupported(String objectModel);
296
297 /**
298  * <p>Set a feature for this <code>XPathFactory</code> and
299  * <code>XPath</code>s created by this factory.</p>
300  *

```

```

299     * <p>
300     * Feature names are fully qualified {@link java.net.URI}s.
301     * Implementations may define their own features.
302     * An {@link XPathFactoryConfigurationException} is thrown if this
303     * <code>XPathFactory</code> or the <code>XPath</code>s
304     * it creates cannot support the feature.
305     * It is possible for an <code>XPathFactory</code> to expose a feature value
306     * but be unable to change its state.
307     * </p>
308     *
309     * <p>
310     * All implementations are required to support the {@link javax.xml.XMLConstants#
311         FEATURE_SECURE_PROCESSING} feature.
312     * When the feature is <code>true</code>, any reference to an external function is an error.
313     * Under these conditions, the implementation must not call the {@link XPathFunctionResolver}
314     * and must throw an {@link XPathFunctionException}.
315     * </p>
316     *
317     * @param name Feature name.
318     * @param value Is feature state <code>true</code> or <code>false</code>.
319     *
320     * @throws XPathFactoryConfigurationException if this <code>XPathFactory</code> or the <code>
321         XPath</code>s
322     * it creates cannot support this feature.
323     * @throws NullPointerException if <code>name</code> is <code>null</code>.
324     */
325     public abstract void setFeature(String name, boolean value)
326         throws XPathFactoryConfigurationException;
327
328     /**
329     * <p>Get the state of the named feature.</p>
330     *
331     * <p>
332     * Feature names are fully qualified {@link java.net.URI}s.
333     * Implementations may define their own features.
334     * An {@link XPathFactoryConfigurationException} is thrown if this
335     * <code>XPathFactory</code> or the <code>XPath</code>s
336     * it creates cannot support the feature.
337     * It is possible for an <code>XPathFactory</code> to expose a feature value
338     * but be unable to change its state.
339     * </p>
340     *
341     * @param name Feature name.
342     *
343     * @return State of the named feature.
344     *
345     * @throws XPathFactoryConfigurationException if this
346     * <code>XPathFactory</code> or the <code>XPath</code>s
347     * it creates cannot support this feature.
348     * @throws NullPointerException if <code>name</code> is <code>null</code>.
349     */
350     public abstract boolean getFeature(String name)
351         throws XPathFactoryConfigurationException;

```



```

350
351 /**
352  * <p>Establish a default variable resolver.</p>
353  *
354  * <p>Any <code>XPath</code> objects constructed from this factory will use
355  * the specified resolver by default.</p>
356  *
357  * <p>A <code>NullPointerException</code> is thrown if <code>resolver</code>
358  * is <code>null</code>.</p>
359  *
360  * @param resolver Variable resolver.
361  *
362  * @throws NullPointerException If <code>resolver</code> is
363  *    <code>null</code>.
364  */
365 public abstract void setXPathVariableResolver(XPathVariableResolver resolver);
366
367 /**
368  * <p>Establish a default function resolver.</p>
369  *
370  * <p>Any <code>XPath</code> objects constructed from this factory will
371  * use the specified resolver by default.</p>
372  *
373  * <p>A <code>NullPointerException</code> is thrown if
374  * <code>resolver</code> is <code>null</code>.</p>
375  *
376  * @param resolver XPath function resolver.
377  *
378  * @throws NullPointerException If <code>resolver</code> is
379  *    <code>null</code>.
380  */
381 public abstract void setXPathFunctionResolver(XPathFunctionResolver resolver);
382
383 /**
384  * <p>Return a new <code>XPath</code> using the underlying object
385  * model determined when the <code>XPathFactory</code> was instantiated.</p>
386  *
387  * @return New instance of an <code>XPath</code>.
388  */
389 public abstract XPath newXPath();
390 }

```

## 35.8 XPathFactoryConfigurationException.java

Secuencia 326: javax/xml/xpath/XPathFactoryConfigurationException.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 /**
29  * <code>XPathFactoryConfigurationException</code> represents a configuration error in a <code>
    XPathFactory</code> environment.</p>
30  *
31  * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
32  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
33  * @since 1.5
34  */
35 public class XPathFactoryConfigurationException extends XPathException {
36
37     /**
38      * <p>Stream Unique Identifier.</p>
39      */
40     private static final long serialVersionUID = -1837080260374986980L;
41
42     /**
43      * <p>Constructs a new <code>XPathFactoryConfigurationException</code> with the specified
        detail <code>message</code>.</p>
44      *
45      * <p>The <code>cause</code> is not initialized.</p>
46      *
47      * <p>If <code>message</code> is <code>null</code>,
48      * then a <code>NullPointerException</code> is thrown.</p>
49      *
50      * @param message The detail message.
51      *
52      * @throws NullPointerException When <code>message</code> is
53      * <code>null</code>.
54      */
55     public XPathFactoryConfigurationException(String message) {
56         super(message);
57     }
58
59     /**
```

```

60      * <p>Constructs a new <code>XPathFactoryConfigurationException</code>
61      * with the specified <code>cause</code>.</p>
62      *
63      * <p>If <code>cause</code> is <code>>null</code>,
64      * then a <code>NullPointerException</code> is thrown.</p>
65      *
66      * @param cause The cause.
67      *
68      * @throws NullPointerException if <code>cause</code> is <code>>null</code>.
69      */
70      public XPathFactoryConfigurationException(Throwable cause) {
71          super(cause);
72      }
73  }

```

### 35.9 XPathFactoryFinder.java

Secuencia 327: javax/xml/xpath/XPathFactoryFinder.java

```

1  /*
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 import java.io.File;
29 import java.lang.reflect.Method;
30 import java.lang.reflect.Modifier;
31 import java.net.URL;
32 import java.security.AccessControlContext;
33 import java.security.AccessController;
34 import java.security.PrivilegedAction;
35 import java.util.Properties;

```

```
36 import java.util.ServiceConfigurationError;
37 import java.util.ServiceLoader;
38
39 /**
40  * Implementation of {@link XPathFactory#newInstance(String)}.
41  *
42  * @author <a href="Kohsuke.Kawaguchi@Sun.com">Kohsuke Kawaguchi</a>
43  * @version $Revision: 1.7 $, $Date: 2010-11-01 04:36:14 $
44  * @since 1.5
45  */
46 class XPathFactoryFinder {
47     private static final String DEFAULT_PACKAGE = "com.sun.org.apache.xpath.internal";
48
49     private static final SecuritySupport ss = new SecuritySupport() ;
50     /** debug support code. */
51     private static boolean debug = false;
52     static {
53         // Use try/catch block to support applets
54         try {
55             debug = ss.getProperty("jaxp.debug") != null;
56         } catch (Exception unused) {
57             debug = false;
58         }
59     }
60
61     /**
62      * <p>Cache properties for performance.</p>
63      */
64     private static final Properties cacheProps = new Properties();
65
66     /**
67      * <p>First time requires initialization overhead.</p>
68      */
69     private volatile static boolean firstTime = true;
70
71     /**
72      * <p>Conditional debug printing.</p>
73      *
74      * @param msg to print
75      */
76     private static void debugPrintln(String msg) {
77         if (debug) {
78             System.err.println("JAXP: " + msg);
79         }
80     }
81
82     /**
83      * <p><code>ClassLoader</code> to use to find <code>XPathFactory</code>.</p>
84      */
85     private final ClassLoader classLoader;
86
87     /**
88      * <p>Constructor that specifies <code>ClassLoader</code> to use
```

```

89      * to find <code>XPathFactory</code>.</p>
90      *
91      * @param loader
92      *     to be used to load resource and {@link XPathFactory}
93      *     implementations during the resolution process.
94      *     If this parameter is null, the default system class loader
95      *     will be used.
96      */
97     public XPathFactoryFinder(ClassLoader loader) {
98         this.classLoader = loader;
99         if( debug ) {
100             debugDisplayClassLoader();
101         }
102     }
103
104     private void debugDisplayClassLoader() {
105         try {
106             if( classLoader == ss.getContextClassLoader() ) {
107                 debugPrintln("using thread context class loader (" + classLoader + ") for search");
108                 return;
109             }
110         } catch( Throwable unused ) {
111             // getContextClassLoader() undefined in JDK1.1
112         }
113
114         if( classLoader == ClassLoader.getSystemClassLoader() ) {
115             debugPrintln("using system class loader (" + classLoader + ") for search");
116             return;
117         }
118
119         debugPrintln("using class loader (" + classLoader + ") for search");
120     }
121
122     /**
123      * <p>Creates a new {@link XPathFactory} object for the specified
124      * object model.</p>
125      *
126      * @param uri
127      *     Identifies the underlying object model.
128      *
129      * @return <code>null</code> if the callee fails to create one.
130      *
131      * @throws NullPointerException
132      *     If the parameter is null.
133      */
134     public XPathFactory newFactory(String uri) throws XPathFactoryConfigurationException {
135         if (uri == null) {
136             throw new NullPointerException();
137         }
138         XPathFactory f = _newFactory(uri);
139         if (f != null) {
140             debugPrintln("factory '" + f.getClass().getName() + "' was found for " + uri);
141         } else {

```

```

142     debugPrintln("unable to find a factory for " + uri);
143 }
144     return f;
145 }
146
147 /**
148  * <p>Lookup a {@link XPathFactory} for the given object model.</p>
149  *
150  * @param uri identifies the object model.
151  *
152  * @return {@link XPathFactory} for the given object model.
153  */
154 private XPathFactory _newFactory(String uri) throws XPathFactoryConfigurationException {
155     XPathFactory xpathFactory = null;
156
157     String propertyName = SERVICE_CLASS.getName() + ":" + uri;
158
159     // system property look up
160     try {
161         debugPrintln("Looking up system property '"+propertyName+"' ");
162         String r = ss.getSystemProperty(propertyName);
163         if(r!=null) {
164             debugPrintln("The value is '"+r+"'");
165             xpathFactory = createInstance(r, true);
166             if (xpathFactory != null) {
167                 return xpathFactory;
168             }
169         } else
170             debugPrintln("The property is undefined.");
171     } catch( Throwable t ) {
172         if( debug ) {
173             debugPrintln("failed to look up system property '"+propertyName+"' ");
174             t.printStackTrace();
175         }
176     }
177
178     String javah = ss.getSystemProperty( "java.home" );
179     String configFile = javah + File.separator +
180     "lib" + File.separator + "jaxp.properties";
181
182     // try to read from $java.home/lib/jaxp.properties
183     try {
184         if(firstTime){
185             synchronized(cacheProps){
186                 if(firstTime){
187                     File f=new File( configFile );
188                     firstTime = false;
189                     if(ss.exists(f)){
190                         debugPrintln("Read properties file " + f);
191                         cacheProps.load(ss.getInputStream(f));
192                     }
193                 }
194             }
195         }
196     }

```

```

195     }
196     final String factoryClassName = cacheProps.getProperty(propertyName);
197     debugPrintln("found " + factoryClassName + " in $java.home/jaxp.properties");
198
199     if (factoryClassName != null) {
200         xpathFactory = createInstance(factoryClassName, true);
201         if(xpathFactory != null){
202             return xpathFactory;
203         }
204     }
205 } catch (Exception ex) {
206     if (debug) {
207         ex.printStackTrace();
208     }
209 }
210
211 // Try with ServiceLoader
212 assert xpathFactory == null;
213 xpathFactory = findServiceProvider(uri);
214
215 // The following assertion should always be true.
216 // Uncomment it, recompile, and run with -ea in case of doubts:
217 // assert xpathFactory == null || xpathFactory.isObjectModelSupported(uri);
218
219 if (xpathFactory != null) {
220     return xpathFactory;
221 }
222
223 // platform default
224 if(uri.equals(XPathFactory.DEFAULT_OBJECT_MODEL_URI)) {
225     debugPrintln("attempting to use the platform default W3C DOM XPath lib");
226     return createInstance("com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl", true);
227 }
228
229 debugPrintln("all things were tried, but none was found. bailing out.");
230 return null;
231 }
232
233 /** <p>Create class using appropriate ClassLoader.</p>
234  *
235  * @param className Name of class to create.
236  * @return Created class or <code>null</code>.
237  */
238 private Class<?> createClass(String className) {
239     Class clazz;
240     // make sure we have access to restricted packages
241     boolean internal = false;
242     if (System.getSecurityManager() != null) {
243         if (className != null && className.startsWith(DEFAULT_PACKAGE)) {
244             internal = true;
245         }
246     }
247

```

```

248     // use appropriate ClassLoader
249     try {
250         if (classLoader != null && !internal) {
251             clazz = Class.forName(className, false, classLoader);
252         } else {
253             clazz = Class.forName(className);
254         }
255     } catch (Throwable t) {
256         if(debug) {
257             t.printStackTrace();
258         }
259         return null;
260     }
261
262     return clazz;
263 }
264
265 /**
266  * <p>Creates an instance of the specified and returns it.</p>
267  *
268  * @param className
269  *     fully qualified class name to be instantiated.
270  *
271  * @return null
272  *     if it fails. Error messages will be printed by this method.
273  */
274 XPathFactory createInstance( String className )
275     throws XPathFactoryConfigurationException
276 {
277     return createInstance( className, false );
278 }
279
280 XPathFactory createInstance( String className, boolean useServicesMechanism )
281     throws XPathFactoryConfigurationException
282 {
283     XPathFactory xPathFactory = null;
284
285     debugPrintln("createInstance(" + className + ")");
286
287     // get Class from className
288     Class<?> clazz = createClass(className);
289     if (clazz == null) {
290         debugPrintln("failed to getClass(" + className + ")");
291         return null;
292     }
293     debugPrintln("loaded " + className + " from " + which(clazz));
294
295     // instantiate Class as a XPathFactory
296     try {
297         if (!useServicesMechanism) {
298             xPathFactory = newInstanceNoServiceLoader(clazz);
299         }
300         if (xPathFactory == null) {

```



```

301         xPathFactory = (XPathFactory) clazz.newInstance();
302     }
303     } catch (ClassCastException classCastException) {
304         debugPrintln("could not instantiate " + clazz.getName());
305         if (debug) {
306             classCastException.printStackTrace();
307         }
308         return null;
309     } catch (IllegalAccessException illegalAccessException) {
310         debugPrintln("could not instantiate " + clazz.getName());
311         if (debug) {
312             illegalAccessException.printStackTrace();
313         }
314         return null;
315     } catch (InstantiationException instantiationException) {
316         debugPrintln("could not instantiate " + clazz.getName());
317         if (debug) {
318             instantiationException.printStackTrace();
319         }
320         return null;
321     }
322
323     return xPathFactory;
324 }
325 /**
326  * Try to construct using newXPathFactoryNoServiceLoader
327  * method if available.
328  */
329 private static XPathFactory newInstanceNoServiceLoader(
330     Class<?> providerClass
331 ) throws XPathFactoryConfigurationException {
332     // Retain maximum compatibility if no security manager.
333     if (System.getSecurityManager() == null) {
334         return null;
335     }
336     try {
337         Method creationMethod =
338             providerClass.getDeclaredMethod(
339                 "newXPathFactoryNoServiceLoader"
340             );
341         final int modifiers = creationMethod.getModifiers();
342
343         // Do not call "newXPathFactoryNoServiceLoader" if it's
344         // not public static.
345         if (!Modifier.isStatic(modifiers) || !Modifier.isPublic(modifiers)) {
346             return null;
347         }
348
349         // Only calls "newXPathFactoryNoServiceLoader" if it's
350         // declared to return an instance of XPathFactory.
351         final Class<?> returnType = creationMethod.getReturnType();
352         if (SERVICE_CLASS.isAssignableFrom(returnType)) {
353             return SERVICE_CLASS.cast(creationMethod.invoke(null, (Object[])null));

```

```

354         } else {
355             // Should not happen since
356             // XPathFactoryImpl.newXPathFactoryNoServiceLoader is
357             // declared to return XPathFactory.
358             throw new ClassCastException(returnType
359                 + " cannot be cast to " + SERVICE_CLASS);
360         }
361     } catch (ClassCastException e) {
362         throw new XPathFactoryConfigurationException(e);
363     } catch (NoSuchMethodException exc) {
364         return null;
365     } catch (Exception exc) {
366         return null;
367     }
368 }
369
370 // Call isObjectModelSupportedBy with initial context.
371 private boolean isObjectModelSupportedBy(final XPathFactory factory,
372     final String objectModel,
373     AccessControlContext acc) {
374     return AccessController.doPrivileged(new PrivilegedAction<Boolean>() {
375         public Boolean run() {
376             return factory.isObjectModelSupported(objectModel);
377         }
378     }, acc);
379 }
380
381 /**
382  * Finds a service provider subclass of XPathFactory that supports the
383  * given object model using the ServiceLoader.
384  *
385  * @param objectModel URI of object model to support.
386  * @return An XPathFactory supporting the specified object model, or null
387  *         if none is found.
388  * @throws XPathFactoryConfigurationException if a configuration error is found.
389  */
390 private XPathFactory findServiceProvider(final String objectModel)
391     throws XPathFactoryConfigurationException {
392
393     assert objectModel != null;
394     // store current context.
395     final AccessControlContext acc = AccessController.getContext();
396     try {
397         return AccessController.doPrivileged(new PrivilegedAction<XPathFactory>() {
398             public XPathFactory run() {
399                 final ServiceLoader<XPathFactory> loader =
400                     ServiceLoader.load(SERVICE_CLASS);
401                 for (XPathFactory factory : loader) {
402                     // restore initial context to call
403                     // factory.isObjectModelSupportedBy
404                     if (isObjectModelSupportedBy(factory, objectModel, acc)) {
405                         return factory;
406                     }

```

```

407         }
408         return null; // no factory found.
409     }
410     });
411     } catch (ServiceConfigurationError error) {
412         throw new XPathFactoryConfigurationException(error);
413     }
414 }
415
416 private static final Class<XPathFactory> SERVICE_CLASS = XPathFactory.class;
417
418 private static String which( Class clazz ) {
419     return which( clazz.getName(), clazz.getClassLoader() );
420 }
421
422 /**
423  * <p>Search the specified classloader for the given classname.</p>
424  *
425  * @param classname the fully qualified name of the class to search for
426  * @param loader the classloader to search
427  *
428  * @return the source location of the resource, or null if it wasn't found
429  */
430 private static String which(String classname, ClassLoader loader) {
431
432     String classnameAsResource = classname.replace('.', '/') + ".class";
433
434     if( loader==null ) loader = ClassLoader.getSystemClassLoader();
435
436     //URL it = loader.getResource(classnameAsResource);
437     URL it = ss.getResourceAsURL(loader, classnameAsResource);
438     if (it != null) {
439         return it.toString();
440     } else {
441         return null;
442     }
443 }
444 }

```

### 35.10 XPathFunction.java

Secuencia 328: javax/xml/xpath/XPathFunction.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *

```

```

12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.xpath;
27
28 import java.util.List;
29
30 /**
31  * <p><code>XPathFunction</code> provides access to XPath functions.</p>
32  *
33  * <p>Functions are identified by QName and arity in XPath.</p>
34  *
35  * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
36  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
37  * @since 1.5
38  */
39 public interface XPathFunction {
40     /**
41      * <p>Evaluate the function with the specified arguments.</p>
42      *
43      * <p>To the greatest extent possible, side-effects should be avoided in the
44      * definition of extension functions. The implementation evaluating an
45      * XPath expression is under no obligation to call extension functions in
46      * any particular order or any particular number of times.</p>
47      *
48      * @param args The arguments, <code>null</code> is a valid value.
49      *
50      * @return The result of evaluating the <code>XPath</code> function as an <code>Object</code>.
51      *
52      * @throws XPathFunctionException If <code>args</code> cannot be evaluated with this <code>XPath
53      * </code> function.
54      */
55     public Object evaluate(List args)
56         throws XPathFunctionException;
57 }

```

### 35.11 XPathFunctionException.java

Secuencia 329: javax/xml/xpath/XPathFunctionException.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

```

```

4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 /**
29  * <code>XPathFunctionException</code> represents an error with an XPath function.</p>
30  *
31  * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
32  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
33  * @since 1.5
34  */
35 public class XPathFunctionException extends XPathExpressionException {
36
37     /**
38      * <p>Stream Unique Identifier.</p>
39      */
40     private static final long serialVersionUID = -1837080260374986980L;
41
42     /**
43      * <p>Constructs a new <code>XPathFunctionException</code> with the specified detail <code>
44         message</code>.</p>
45      *
46      * <p>The <code>cause</code> is not initialized.</p>
47      *
48      * <p>If <code>message</code> is <code>>null</code>,
49      * then a <code>NullPointerException</code> is thrown.</p>
50      *
51      * @param message The detail message.
52      *
53      * @throws NullPointerException When <code>message</code> is
54      * <code>>null</code>.
55      */
56     public XPathFunctionException(String message) {

```

```

56     super(message);
57 }
58
59 /**
60  * <p>Constructs a new <code>XPathFunctionException</code> with the specified <code>cause</code>
61  * </p>
62  * <p>If <code>cause</code> is <code>null</code>,
63  * then a <code>NullPointerException</code> is thrown.</p>
64  *
65  * @param cause The cause.
66  *
67  * @throws NullPointerException if <code>cause</code> is <code>null</code>.
68  */
69 public XPathFunctionException(Throwable cause) {
70     super(cause);
71 }
72 }

```

### 35.12 XPathFunctionResolver.java

Secuencia 330: javax.xml.xpath/XPathFunctionResolver.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package javax.xml.xpath;
27
28 import javax.xml.namespace.QName;
29
30 /**

```

```

31 * <p><code>XPathFunctionResolver</code> provides access to the set of user defined <code>
    XPathFunction</code>s.</p>
32 *
33 * <p>XPath functions are resolved by name and arity.
34 * The resolver is not needed for XPath built-in functions and the resolver
35 * <strong><em>cannot</em></strong> be used to override those functions.</p>
36 *
37 * <p>In particular, the resolver is only called for functions in an another
38 * namespace (functions with an explicit prefix). This means that you cannot
39 * use the <code>XPathFunctionResolver</code> to implement specifications
40 * like <a href="http://www.w3.org/TR/xmlsig-core/">XML-Signature Syntax
41 * and Processing</a> which extend the function library of XPath 1.0 in the
42 * same namespace. This is a consequence of the design of the resolver.</p>
43 *
44 * <p>If you wish to implement additional built-in functions, you will have to
45 * extend the underlying implementation directly.</p>
46 *
47 * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
48 * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
49 * @see <a href="http://www.w3.org/TR/xpath#corelib">XML Path Language (XPath) Version 1.0, Core
    Function Library</a>
50 * @since 1.5
51 */
52 public interface XPathFunctionResolver {
53     /**
54      * <p>Find a function in the set of available functions.</p>
55      *
56      * <p>If <code>functionName</code> or <code>arity</code> is <code>null</code>, then a <code>
        NullPointerException</code> is thrown.</p>
57      *
58      * @param functionName The function name.
59      * @param arity The number of arguments that the returned function must accept.
60      *
61      * @return The function or <code>null</code> if no function named <code>functionName</code> with
        <code>arity</code> arguments exists.
62      *
63      * @throws NullPointerException If <code>functionName</code> or <code>arity</code> is <code>null
        </code>.
64      */
65     public XPathFunction resolveFunction(QName functionName, int arity);
66 }

```

### 35.13 XPathVariableResolver.java

Secuencia 331: javax/xml/xpath/XPathVariableResolver.java

```

1  /*
2  * Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```

 9  *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 package javax.xml.xpath;
27
28 import javax.xml.namespace.QName;
29
30 /**
31  * <p><code>XPathVariableResolver</code> provides access to the set of user defined XPath variables
32  * .</p>
33  *
34  * <p>The <code>XPathVariableResolver</code> and the XPath evaluator must adhere to a contract that
35  * cannot be directly enforced by the API. Although variables may be mutable,
36  * that is, an application may wish to evaluate the same XPath expression more
37  * than once with different variable values, in the course of evaluating any
38  * single XPath expression, a variable's value <strong><em>must</em></strong>
39  * not change.</p>
40  *
41  * @author <a href="mailto:Norman.Walsh@Sun.com">Norman Walsh</a>
42  * @author <a href="mailto:Jeff.Suttor@Sun.com">Jeff Suttor</a>
43  * @since 1.5
44  */
45 public interface XPathVariableResolver {
46     /**
47      * <p>Find a variable in the set of available variables.</p>
48      *
49      * <p>If <code>variableName</code> is <code>>null</code>, then a <code>NullPointerException</code>
50      * is thrown.</p>
51      *
52      * @param variableName The <code>QName</code> of the variable name.
53      *
54      * @return The variables value, or <code>>null</code> if no variable named <code>variableName</code>
55      * exists. The value returned must be of a type appropriate for the underlying object model.
56      *
57      * @throws NullPointerException If <code>variableName</code> is <code>>null</code>.
58      */
59     public Object resolveVariable(QName variableName);
60 }

```



## 36 Dom (org.w3c.dom package)

### 36.1 Attr.java

Secuencia 332: org.w3c.dom/Attr.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45 * The Attr interface represents an attribute in an
46 * Element object. Typically the allowable values for the
47 * attribute are defined in a schema associated with the document.

```

```
48 * <p><code>Attr</code> objects inherit the <code>Node</code> interface, but
49 * since they are not actually child nodes of the element they describe, the
50 * DOM does not consider them part of the document tree. Thus, the
51 * <code>Node</code> attributes <code>parentNode</code>,
52 * <code>previousSibling</code>, and <code>nextSibling</code> have a
53 * <code>null</code> value for <code>Attr</code> objects. The DOM takes the
54 * view that attributes are properties of elements rather than having a
55 * separate identity from the elements they are associated with; this should
56 * make it more efficient to implement such features as default attributes
57 * associated with all elements of a given type. Furthermore,
58 * <code>Attr</code> nodes may not be immediate children of a
59 * <code>DocumentFragment</code>. However, they can be associated with
60 * <code>Element</code> nodes contained within a
61 * <code>DocumentFragment</code>. In short, users and implementors of the
62 * DOM need to be aware that <code>Attr</code> nodes have some things in
63 * common with other objects inheriting the <code>Node</code> interface, but
64 * they also are quite distinct.
65 * <p>The attribute's effective value is determined as follows: if this
66 * attribute has been explicitly assigned any value, that value is the
67 * attribute's effective value; otherwise, if there is a declaration for
68 * this attribute, and that declaration includes a default value, then that
69 * default value is the attribute's effective value; otherwise, the
70 * attribute does not exist on this element in the structure model until it
71 * has been explicitly added. Note that the <code>Node.nodeValue</code>
72 * attribute on the <code>Attr</code> instance can also be used to retrieve
73 * the string version of the attribute's value(s).
74 * <p> If the attribute was not explicitly given a value in the instance
75 * document but has a default value provided by the schema associated with
76 * the document, an attribute node will be created with
77 * <code>specified</code> set to <code>false</code>. Removing attribute
78 * nodes for which a default value is defined in the schema generates a new
79 * attribute node with the default value and <code>specified</code> set to
80 * <code>false</code>. If validation occurred while invoking
81 * <code>Document.normalizeDocument()</code>, attribute nodes with
82 * <code>specified</code> equals to <code>false</code> are recomputed
83 * according to the default attribute values provided by the schema. If no
84 * default value is associate with this attribute in the schema, the
85 * attribute node is discarded.
86 * <p>In XML, where the value of an attribute can contain entity references,
87 * the child nodes of the <code>Attr</code> node may be either
88 * <code>Text</code> or <code>EntityReference</code> nodes (when these are
89 * in use; see the description of <code>EntityReference</code> for
90 * discussion).
91 * <p>The DOM Core represents all attribute values as simple strings, even if
92 * the DTD or schema associated with the document declares them of some
93 * specific type such as tokenized.
94 * <p>The way attribute value normalization is performed by the DOM
95 * implementation depends on how much the implementation knows about the
96 * schema in use. Typically, the <code>value</code> and
97 * <code>nodeValue</code> attributes of an <code>Attr</code> node initially
98 * returns the normalized value given by the parser. It is also the case
99 * after <code>Document.normalizeDocument()</code> is called (assuming the
100 * right options have been set). But this may not be the case after
```

```

101 * mutation, independently of whether the mutation is performed by setting
102 * the string value directly or by changing the Attr child
103 * nodes. In particular, this is true when  are involved, given that they are not represented in the DOM and they
105 \* impact attribute value normalization. On the other hand, if the
106 \* implementation knows about the schema in use when the attribute value is
107 \* changed, and it is of a different type than CDATA, it may normalize it
108 \* again at that time. This is especially true of specialized DOM
109 \* implementations, such as SVG DOM implementations, which store attribute
110 \* values in an internal form different from a string.
111 \* <p>The following table gives some examples of the relations between the
112 \* attribute value in the original document \(parsed attribute\), the value as
113 \* exposed in the DOM, and the serialization of the value:
114 \* <table border='1' cellpadding='3'>
115 \* <tr>
116 \* <th>Examples</th>
117 \* <th>Parsed
118 \* attribute value</th>
119 \* <th>Initial Attr.value</th>
120 \* <th>Serialized attribute value</th>
121 \* </tr>
122 \* <tr>
123 \* <td valign='top' rowspan='1' colspan='1'>
124 \* Character reference</td>
125 \* <td valign='top' rowspan='1' colspan='1'>
126 \* <pre>"x&#178;=5"</pre>
127 \* </td>
128 \* <td valign='top' rowspan='1' colspan='1'>
129 \* <pre>"x\u00b2=5"</pre>
130 \* </td>
131 \* <td valign='top' rowspan='1' colspan='1'>
132 \* <pre>"x&#178;=5"</pre>
133 \* </td>
134 \* </tr>
135 \* <tr>
136 \* <td valign='top' rowspan='1' colspan='1'>Built-in
137 \* character entity</td>
138 \* <td valign='top' rowspan='1' colspan='1'>
139 \* <pre>"y&lt;6"</pre>
140 \* </td>
141 \* <td valign='top' rowspan='1' colspan='1'>
142 \* <pre>"y&lt;6"</pre>
143 \* </td>
144 \* <td valign='top' rowspan='1' colspan='1'>
145 \* <pre>"y&lt;6"</pre>
146 \* </td>
147 \* </tr>
148 \* <tr>
149 \* <td valign='top' rowspan='1' colspan='1'>Literal newline between</td>
150 \* <td valign='top' rowspan='1' colspan='1'>
151 \* <pre>
152 \* "x=5&#10;y=6"</pre>

```

```

153 * </td>
154 * <td valign='top' rowspan='1' colspan='1'>
155 * <pre>"x=5 y=6"</pre>
156 * </td>
157 * <td valign='top' rowspan='1' colspan='1'>
158 * <pre>"x=5&#10;y=6"</pre>
159 * </td>
160 * </tr>
161 * <tr>
162 * <td valign='top' rowspan='1' colspan='1'>Normalized newline between</td>
163 * <td valign='top' rowspan='1' colspan='1'>
164 * <pre>"x=5
165 * y=6"</pre>
166 * </td>
167 * <td valign='top' rowspan='1' colspan='1'>
168 * <pre>"x=5 y=6"</pre>
169 * </td>
170 * <td valign='top' rowspan='1' colspan='1'>
171 * <pre>"x=5 y=6"</pre>
172 * </td>
173 * </tr>
174 * <tr>
175 * <td valign='top' rowspan='1' colspan='1'>Entity <code>e</code> with literal newline</td>
176 * <td valign='top' rowspan='1' colspan='1'>
177 * <pre>
178 * &lt;!ENTITY e '...&#10;...&#10;'> [...]&#10; "x=5&#10;y=6"</pre>
179 * </td>
180 * <td valign='top' rowspan='1' colspan='1'><em>Dependent on Implementation and Load Options</em></td>
181 * <td valign='top' rowspan='1' colspan='1'><em>Dependent on Implementation and Load/Save Options</em></td>
182 * </tr>
183 * </table>
184 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
185 */
186 public interface Attr extends Node {
187     /**
188      * Returns the name of this attribute. If <code>Node.localName</code> is
189      * different from <code>null</code>, this attribute is a qualified name.
190      */
191     public String getName();
192
193     /**
194      * <code>True</code> if this attribute was explicitly given a value in
195      * the instance document, <code>false</code> otherwise. If the
196      * application changed the value of this attribute node (even if it ends
197      * up having the same value as the default value) then it is set to
198      * <code>true</code>. The implementation may handle attributes with
199      * default values from other schemas similarly but applications should
200      * use <code>Document.normalizeDocument()</code> to guarantee this
201      * information is up-to-date.
202      */

```

```
203 public boolean getSpecified();
204
205 /**
206  * On retrieval, the value of the attribute is returned as a string.
207  * Character and general entity references are replaced with their
208  * values. See also the method getAttribute() on the
209  * Element interface.
210  *   
On setting, this creates a Text node with the unparsed
211  * contents of the string, i.e. any characters that an XML processor
212  * would recognize as markup are instead treated as literal text. See
213  * also the method Element.setAttribute().
214  *   
Some specialized implementations, such as some [\]
215  \* implementations, may do normalization automatically, even after
216  \* mutation; in such case, the value on retrieval may differ from the
217  \* value on setting.
218  \*/
219 public String getValue\(\);
220 /\*\*
221  \* On retrieval, the value of the attribute is returned as a string.
222  \* Character and general entity references are replaced with their
223  \* values. See also the method getAttribute\(\) on the
224  \* Element interface.
225  \*   
On setting, this creates a Text node with the unparsed
226  \* contents of the string, i.e. any characters that an XML processor
227  \* would recognize as markup are instead treated as literal text. See
228  \* also the method Element.setAttribute\(\).
229  \*   
Some specialized implementations, such as some \[\\]
230  \\* implementations, may do normalization automatically, even after
231  \\* mutation; in such case, the value on retrieval may differ from the
232  \\* value on setting.
233  \\* @exception DOMException
234  \\*   NO\\_MODIFICATION\\_ALLOWED\\_ERR: Raised when the node is readonly.
235  \\*/
236 public void setValue\\(String value\\)
237     throws DOMException;
238
239 /\\*\\*
240  \\* The Element node this attribute is attached to or
241  \\* null if this attribute is not in use.
242  \\* @since DOM Level 2
243  \\*/
244 public Element getOwnerElement\\(\\);
245
246 /\\*\\*
247  \\* The type information associated with this attribute. While the type
248  \\* information contained in this attribute is guarantee to be correct
249  \\* after loading the document or invoking
250  \\* Document.normalizeDocument\\(\\), schemaTypeInfo
251  \\* may not be reliable if the node was moved.
252  \\* @since DOM Level 3
253  \\*/
```

```

254 public TypeInfo getSchemaTypeInfo();
255
256 /**
257  * Returns whether this attribute is known to be of type ID (i.e. to
258  * contain an identifier for its owner element) or not. When it is and
259  * its value is unique, the ownerElement of this attribute
260  * can be retrieved using the method Document.getElementById
261  * . The implementation could use several ways to determine if an
262  * attribute node is known to contain an identifier:
263  * <ul>
264  * <li> If validation
265  * occurred using an XML Schema [Document.normalizeDocument\(\), the post-schema-validation
269  \* infoSet contributions \(PSVI contributions\) values are used to
270  \* determine if this attribute is a schema-determined ID attribute using
271  \* the Document.normalizeDocument\\(\\), the infoSet b\\[type definition\\]</
278  \\* b> value is used to determine if this attribute is a DTD-determined ID
279  \\* attribute using the Element.setIdAttribute\\\(\\\),
285  \\\* Element.setIdAttributeNS\\\(\\\), or
286  \\\* Element.setIdAttributeNode\\\(\\\), i.e. it is an
287  \\\* user-determined ID attribute;
288  \\\* <p><b>Note:</b> XPointer framework \\\(see section 3.2 in \\\[Document.normalizeDocument\\\\(\\\\), all user-determined ID
301  \\\\* attributes are reset and all attribute nodes ID information are then
302  \\\\* reevaluated in accordance to the schema used. As a consequence, if
303  \\\\* the Attr.schemaTypeInfo attribute contains an ID type,
304  \\\\* isId will always return true.
305  \\\\* @since DOM Level 3

```

```
301     */
302     public boolean isId();
303
304 }
```

## 36.2 CDATASection.java

Secuencia 333: org/w3c/dom/CDATASection.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45  * CDATA sections are used to escape blocks of text containing characters that
```

```

46 * would otherwise be regarded as markup. The only delimiter that is
47 * recognized in a CDATA section is the "]]>" string that ends the CDATA
48 * section. CDATA sections cannot be nested. Their primary purpose is for
49 * including material such as XML fragments, without needing to escape all
50 * the delimiters.
51 * <p>The <code>CharacterData.data</code> attribute holds the text that is
52 * contained by the CDATA section. Note that this <em>may</em> contain characters that need to be
    escaped outside of CDATA sections and
53 * that, depending on the character encoding ("charset") chosen for
54 * serialization, it may be impossible to write out some characters as part
55 * of a CDATA section.
56 * <p>The <code>CDATASection</code> interface inherits from the
57 * <code>CharacterData</code> interface through the <code>Text</code>
58 * interface. Adjacent <code>CDATASection</code> nodes are not merged by use
59 * of the <code>normalize</code> method of the <code>Node</code> interface.
60 * <p> No lexical check is done on the content of a CDATA section and it is
61 * therefore possible to have the character sequence <code>"]]>"</code>
62 * in the content, which is illegal in a CDATA section per section 2.7 of [

```

### 36.3 CharacterData.java

Secuencia 334: org/w3c/dom/CharacterData.java

```

1 /*
2 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3 *
4 *
5 *
6 *
7 *
8 *

```



```

 9  *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42 package org.w3c.dom;
43
44 /**
45  * The CharacterData interface extends Node with a set of
46  * attributes and methods for accessing character data in the DOM. For
47  * clarity this set is defined here rather than on each object that uses
48  * these attributes and methods. No DOM objects correspond directly to
49  * CharacterData, though Text and others do
50  * inherit the interface from it. All offsets in this interface
51  * start from 0.
52  * 

As explained in the DOMString interface, text strings in
53  * the DOM are represented in UTF-16, i.e. as a sequence of 16-bit units. In
54  * the following, the term 16-bit units is used whenever necessary to
55  * indicate that indexing on CharacterData is done in 16-bit units.
56  * 

See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 Document
57  * Object Model (DOM) Level 3 Core Specification.
58  */
59
60 public interface CharacterData extends Node {
61     /**
62      * The character data of the node that implements this interface. The DOM


```

```

61     * implementation may not put arbitrary limits on the amount of data
62     * that may be stored in a <code>CharacterData</code> node. However,
63     * implementation limits may mean that the entirety of a node's data may
64     * not fit into a single <code>DOMString</code>. In such cases, the user
65     * may call <code>substringData</code> to retrieve the data in
66     * appropriately sized pieces.
67     * @exception DOMException
68     *     DOMSTRING_SIZE_ERR: Raised when it would return more characters than
69     *     fit in a <code>DOMString</code> variable on the implementation
70     *     platform.
71     */
72     public String getData()
73         throws DOMException;
74
75     /**
76     * The character data of the node that implements this interface. The DOM
77     * implementation may not put arbitrary limits on the amount of data
78     * that may be stored in a <code>CharacterData</code> node. However,
79     * implementation limits may mean that the entirety of a node's data may
80     * not fit into a single <code>DOMString</code>. In such cases, the user
81     * may call <code>substringData</code> to retrieve the data in
82     * appropriately sized pieces.
83     * @exception DOMException
84     *     NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.
85     */
86     public void setData(String data)
87         throws DOMException;
88
89     /**
90     * The number of 16-bit units that are available through <code>data</code>
91     * and the <code>substringData</code> method below. This may have the
92     * value zero, i.e., <code>CharacterData</code> nodes may be empty.
93     */
94     public int getLength();
95
96     /**
97     * Extracts a range of data from the node.
98     * @param offset Start offset of substring to extract.
99     * @param count The number of 16-bit units to extract.
100    * @return The specified substring. If the sum of <code>offset</code> and
101    *     <code>count</code> exceeds the <code>length</code>, then all 16-bit
102    *     units to the end of the data are returned.
103    * @exception DOMException
104    *     INDEX_SIZE_ERR: Raised if the specified <code>offset</code> is
105    *     negative or greater than the number of 16-bit units in
106    *     <code>data</code>, or if the specified <code>count</code> is
107    *     negative.
108    *     <br>DOMSTRING_SIZE_ERR: Raised if the specified range of text does
109    *     not fit into a <code>DOMString</code>.
110    */
111    public String substringData(int offset,
112                                int count)
113        throws DOMException;

```

```

114 /**
115  * Append the string to the end of the character data of the node. Upon
116  * success, data provides access to the concatenation of
117  * data and the DOMString specified.
118  * @param arg The DOMString to append.
119  * @exception DOMException
120  *   NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
121  */
122 public void appendData(String arg)
123     throws DOMException;
124
125 /**
126  * Insert a string at the specified 16-bit unit offset.
127  * @param offset The character offset at which to insert.
128  * @param arg The DOMString to insert.
129  * @exception DOMException
130  *   INDEX_SIZE_ERR: Raised if the specified offset is
131  *   negative or greater than the number of 16-bit units in
132  *   data.
133  *   NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
134  */
135 public void insertData(int offset,
136     String arg)
137     throws DOMException;
138
139 /**
140  * Remove a range of 16-bit units from the node. Upon success,
141  * data and length reflect the change.
142  * @param offset The offset from which to start removing.
143  * @param count The number of 16-bit units to delete. If the sum of
144  *   offset and count exceeds
145  *   length then all 16-bit units from offset
146  *   to the end of the data are deleted.
147  * @exception DOMException
148  *   INDEX_SIZE_ERR: Raised if the specified offset is
149  *   negative or greater than the number of 16-bit units in
150  *   data, or if the specified count is
151  *   negative.
152  *   NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
153  */
154 public void deleteData(int offset,
155     int count)
156     throws DOMException;
157
158 /**
159  * Replace the characters starting at the specified 16-bit unit offset
160  * with the specified string.
161  * @param offset The offset from which to start replacing.
162  * @param count The number of 16-bit units to replace. If the sum of
163  *   offset and count exceeds
164  *   length, then all 16-bit units to the end of the data
165  *   are replaced; (i.e., the effect is the same as a remove
166  *   method call with the same range, followed by an append

```

```

167     *   method invocation).
168     * @param arg The <code>DOMString</code> with which the range must be
169     *   replaced.
170     * @exception DOMException
171     *   INDEX_SIZE_ERR: Raised if the specified <code>offset</code> is
172     *   negative or greater than the number of 16-bit units in
173     *   <code>data</code>, or if the specified <code>count</code> is
174     *   negative.
175     *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
176     */
177     public void replaceData(int offset,
178                             int count,
179                             String arg)
180         throws DOMException;
181
182 }

```

### 36.4 Comment.java

Secuencia 335: org/w3c/dom/Comment.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for

```

```

34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * This interface inherits from CharacterData and represents the
46   * content of a comment, i.e., all the characters between the starting '
47   * <code>&lt;!--</code>' and ending '<code>--&gt;</code>'. Note that this is
48   * the definition of a comment in XML, and, in practice, HTML, although some
49   * HTML tools may implement the full SGML comment structure.
50   * 

No lexical check is done on the content of a comment and it is
51   * therefore possible to have the character sequence <code>"--</code>
52   * (double-hyphen) in the content, which is illegal in a comment per section
53   * 2.5 of [http://www.w3.org/TR/2004/REC-xml-20040204 XML 1.0</a>]. The
54   * presence of this character sequence must generate a fatal error during
55   * serialization.
56   * 

See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 Document
57   * Object Model (DOM) Level 3 Core Specification</a>.
58   */
59  public interface Comment extends CharacterData {
60  }


```

## 36.5 DOMConfiguration.java

Secuencia 336: org/w3c/dom/DOMConfiguration.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *

```

```

23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * The DOMConfiguration interface represents the configuration
46   * of a document and maintains a table of recognized parameters. Using the
47   * configuration, it is possible to change
48   * Document.normalizeDocument() behavior, such as replacing the
49   * CDATASection nodes with Text nodes or
50   * specifying the type of the schema that must be used when the validation
51   * of the Document is requested. DOMConfiguration
52   * objects are also used in [http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407] DOM
53   * Level 3 Load and Save]
54   * in the DOMParser and DOMSerializer interfaces.
55   * 

The parameter names used by the DOMConfiguration object
56   * are defined throughout the DOM Level 3 specifications. Names are
57   * case-insensitive. To avoid possible conflicts, as a convention, names
58   * referring to parameters defined outside the DOM specification should be
59   * made unique. Because parameters are exposed as properties in names
60   * are recommended to follow the section 5.16 Identifiers of [Unicode] with the addition of the
61   * character '-' (HYPHEN-MINUS) but it is not
62   * enforced by the DOM implementation. DOM Level 3 Core Implementations are
63   * required to recognize all parameters defined in this specification. Some
64   * parameter values may also be required to be supported by the
65   * implementation. Refer to the definition of the parameter to know if a
66   * value must be supported or not.
67   * 

Note: Parameters are similar to features and properties used in
68   * SAX2 [http://www.saxproject.org/] SAX].
69   * 

The following list of parameters defined in the DOM:
70   * 


71   * - "canonical-form"

72   * - 

73   * - true


```

```

74 * <dd><em>optional</em> Canonicalize the document according to the rules specified in [

```

```

123 * </dl></dd>
124 * <dt>
125 * <code>"datatype-normalization"</code></dt>
126 * <dd>
127 * <dl>
128 * <dt><code>true</code></dt>
129 * <dd>[<em>optional</em>] Expose schema normalized values in the tree, such as <a href='http://www
    .w3.org/TR/2001/REC-xmlschema-1-20010502/#key-nv'>XML
130 * Schema normalized values</a> in the case of XML Schema. Since this parameter requires to have
    schema
131 * information, the "validate" parameter will also be set to
132 * <code>true</code>. Having this parameter activated when "validate" is
133 * <code>false</code> has no effect and no schema-normalization will happen.
134 * <p><b>Note:</b> Since the document contains the result of the XML 1.0
135 * processing, this parameter does not apply to attribute value
136 * normalization as defined in section 3.3.3 of [<a href='http://www.w3.org/TR/2004/REC-xml
    -20040204'>XML 1.0</a>] and is only
137 * meant for schema languages other than Document Type Definition (DTD). </dd>
138 * <dt>
139 * <code>false</code></dt>
140 * <dd>[<em>required</em>] (<em>default</em>) Do not perform schema normalization on the tree. </dd>
141 * </dl></dd>
142 * <dt>
143 * <code>"element-content-whitespace"</code></dt>
144 * <dd>
145 * <dl>
146 * <dt><code>true</code></dt>
147 * <dd>[<em>required</em>] (<em>default</em>)Keep all whitespaces in the document.</dd>
148 * <dt><code>false</code></dt>
149 * <dd>[<em>optional</em>] Discard all <code>Text</code> nodes that contain whitespaces in element
150 * content, as described in <a href='http://www.w3.org/TR/2004/REC-xml-infoset-20040204#infoitem.
    character'>
151 * [element content whitespace]</a>. The implementation is expected to use the attribute
152 * <code>Text.isElementContentWhitespace</code> to determine if a
153 * <code>Text</code> node should be discarded or not.</dd>
154 * </dl></dd>
155 * <dt><code>"entities"</code></dt>
156 * <dd>
157 * <dl>
158 * <dt>
159 * <code>true</code></dt>
160 * <dd>[<em>required</em>] (<em>default</em>)Keep <code>EntityReference</code> nodes in the
    document.</dd>
161 * <dt>
162 * <code>false</code></dt>
163 * <dd>[<em>required</em>] Remove all <code>EntityReference</code> nodes from the document,
164 * putting the entity expansions directly in their place. <code>Text</code>
165 * nodes are normalized, as defined in <code>Node.normalize</code>. Only <a href='http://www.w3.org
    /TR/2004/REC-xml-infoset-20040204/#infoitem.rse'>
166 * unexpanded entity references</a> are kept in the document. </dd>
167 * </dl>
168 * <p><b>Note:</b> This parameter does not affect <code>Entity</code> nodes. </dd>

```



```

169 * <dt>
170 * <code>"error-handler"</code></dt>
171 * <dd>[<em>required</em>] Contains a <code>DOMErrorHandler</code> object. If an error is
172 * encountered in the document, the implementation will call back the
173 * <code>DOMErrorHandler</code> registered using this parameter. The
174 * implementation may provide a default <code>DOMErrorHandler</code> object.
175 * When called, <code>DOMError.relatedData</code> will contain the closest
176 * node to where the error occurred. If the implementation is unable to
177 * determine the node where the error occurs,
178 * <code>DOMError.relatedData</code> will contain the <code>Document</code>
179 * node. Mutations to the document from within an error handler will result
180 * in implementation dependent behavior. </dd>
181 * <dt><code>"infoSet"</code></dt>
182 * <dd>
183 * <dl>
184 * <dt>
185 * <code>true</code></dt>
186 * <dd>[<em>required</em>]Keep in the document the information defined in the XML Information Set
187 *   [

```

```

219 * ./dd>
220 * <dt><code>>false</code></dt>
221 * <dd>[<em>required</em>]Discard all namespace declaration attributes. The namespace prefixes (
222 * <code>Node.prefix</code>) are retained even if this parameter is set to
223 * <code>>false</code>.</dd>
224 * </dl></dd>
225 * <dt><code>"normalize-characters"</code></dt>
226 * <dd>
227 * <dl>
228 * <dt><code>true</code></dt>
229 * <dd>[<em>optional</em>] <a href='http://www.w3.org/TR/2004/REC-xml11-20040204/#dt-fullnorm'>
    Fully
230 * normalized</a> the characters in the document as defined in appendix B of [<a href='http://www.
    w3.org/TR/2004/REC-xml11-20040204/'>XML 1.1</a>]. </dd>
231 * <dt>
232 * <code>>false</code></dt>
233 * <dd>[<em>required</em>] (<em>default</em>)Do not perform character normalization.</dd>
234 * </dl></dd>
235 * <dt><code>"schema-location"</code></dt>
236 * <dd>[<em>optional</em>] Represent a <code>DOMString</code> object containing a list of URIs,
237 * separated by whitespaces (characters matching the <a href='http://www.w3.org/TR/2004/REC-xml
    -20040204#NT-S'>nonterminal
238 * production S</a> defined in section 2.3 [<a href='http://www.w3.org/TR/2004/REC-xml-20040204'>
    XML 1.0</a>]), that
239 * represents the schemas against which validation should occur, i.e. the
240 * current schema. The types of schemas referenced in this list must match
241 * the type specified with <code>schema-type</code>, otherwise the behavior
242 * of an implementation is undefined. The schemas specified using this
243 * property take precedence to the schema information specified in the
244 * document itself. For namespace aware schema, if a schema specified using
245 * this property and a schema specified in the document instance (i.e. using
246 * the <code>schemaLocation</code> attribute) in a schema document (i.e.
247 * using schema <code>import</code> mechanisms) share the same
248 * <code>targetNamespace</code>, the schema specified by the user using this
249 * property will be used. If two schemas specified using this property share
250 * the same <code>targetNamespace</code> or have no namespace, the behavior
251 * is implementation dependent. If no location has been provided, this
252 * parameter is <code>null</code>.
253 * <p><b>Note:</b> The <code>"schema-location"</code> parameter is ignored
254 * unless the "schema-type" parameter value is set. It is strongly
255 * recommended that <code>Document.documentURI</code> will be set so that an
256 * implementation can successfully resolve any external entities referenced. </dd>
257 * <dt>
258 * <code>"schema-type"</code></dt>
259 * <dd>[<em>optional</em>] Represent a <code>DOMString</code> object containing an absolute URI
260 * and representing the type of the schema language used to validate a
261 * document against. Note that no lexical checking is done on the absolute
262 * URI. If this parameter is not set, a default value may be provided by
263 * the implementation, based on the schema languages supported and on the
264 * schema language used at load time. If no value is provided, this
265 * parameter is <code>null</code>.
266 * <p><b>Note:</b> For XML Schema [<a href='http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/'>
    XML Schema Part 1</a>]

```

```

267 * , applications must use the value
268 * <code>"http://www.w3.org/2001/XMLSchema"</code>. For XML DTD [false</code></dt>
286 \* <dd>\[<em>required</em>\]Signal an error if a <code>CDATASection</code> contains an
287 \* unrepresentable character.</dd>
288 \* </dl></dd>
289 \* <dt><code>"validate"</code></dt>
290 \* <dd>
291 \* <dl>
292 \* <dt><code>true</code></dt>
293 \* <dd>\[<em>optional</em>\] Require the validation against a schema \(i.e. XML schema, DTD, any
294 \* other type or representation of schema\) of the document as it is being
295 \* normalized as defined by \[false</code>, as specified in
303 \\* the description of the <code>Attr</code> interface;
304 \\* </li>
305 \\* <li>The value of the
306 \\* attribute <code>Text.isElementContentWhitespace</code> for all
307 \\* <code>Text</code> nodes;
308 \\* </li>
309 \\* <li>The value of the attribute
310 \\* <code>Attr.isId</code> for all <code>Attr</code> nodes;
311 \\* </li>
312 \\* <li>The attributes
313 \\* <code>Element.schemaTypeInfo</code> and <code>Attr.schemaTypeInfo</code>.
314 \\* </li>
315 \\* </ul>
316 \\* <p><b>Note:</b> "validate-if-schema" and "validate" are mutually
317 \\* exclusive, setting one of them to <code>true</code> will set the other

```

```

318 * one to false. Applications should also consider setting the
319 * parameter "well-formed" to true, which is the default for
320 * that option, when validating the document. </dd>
321 * <dt>false</dt>
322 * <dd>[required] (default) Do not accomplish schema processing, including the
    internal subset
323 * processing. Default attribute values information are kept. Note that
324 * validation might still happen if "validate-if-schema" is true
325 * . </dd>
326 * </dl></dd>
327 * <dt>"validate-if-schema"</dt>
328 * <dd>
329 * <dl>
330 * <dt>true</dt>
331 * <dd>[optional] Enable validation only if a declaration for the document element can be
332 * found in a schema (independently of where it is found, i.e. XML schema,
333 * DTD, or any other type or representation of schema). If validation is
334 * enabled, this parameter has the same behavior as the parameter "validate"
335 * set to true.
336 * <p><b>Note:</b> "validate-if-schema" and "validate" are mutually
337 * exclusive, setting one of them to true will set the other
338 * one to false. </dd>
339 * <dt>false</dt>
340 * <dd>[required] (default) No schema processing should be performed if the
    document has a schema,
341 * including internal subset processing. Default attribute values
342 * information are kept. Note that validation must still happen if "validate
343 * " is true. </dd>
344 * </dl></dd>
345 * <dt>"well-formed"</dt>
346 * <dd>
347 * <dl>
348 * <dt>true</dt>
349 * <dd>[required] (default) Check if all nodes are XML well formed according to
    the XML version in
350 * use in Document.xmlVersion:
351 * <ul>
352 * <li> check if the attribute
353 * <code>Node.nodeName</code> contains invalid characters according to its
354 * node type and generate a DOMError of type
355 * <code>"wf-invalid-character-in-node-name"</code>, with a
356 * <code>DOMError.SEVERITY_ERROR</code> severity, if necessary;
357 * </li>
358 * <li> check if
359 * the text content inside Attr, Element,
360 * Comment, Text, CDATASection nodes
361 * for invalid characters and generate a DOMError of type
362 * <code>"wf-invalid-character"</code>, with a
363 * <code>DOMError.SEVERITY_ERROR</code> severity, if necessary;
364 * </li>
365 * <li> check if
366 * the data inside ProcessingInstruction nodes for invalid
367 * characters and generate a DOMError of type

```

```

368 * <code>"wf-invalid-character"</code>, with a
369 * <code>DOMError.SEVERITY_ERROR</code> severity, if necessary;
370 * </li>
371 * </ul></dd>
372 * <dt>
373 * <code>>false</code></dt>
374 * <dd><em>optional</em> Do not check for XML well-formedness. </dd>
375 * </dl></dd>
376 * </dl>
377 * <p> The resolution of the system identifiers associated with entities is
378 * done using <code>Document.documentURI</code>. However, when the feature
379 * "LS" defined in [

```

```

419         throws DOMException;
420
421     /**
422     * Check if setting a parameter to a specific value is supported.
423     * @param name The name of the parameter to check.
424     * @param value An object. if null, the returned value is
425     *   true.
426     * @return true if the parameter could be successfully set
427     *   to the specified value, or false if the parameter is
428     *   not recognized or the requested value is not supported. This does
429     *   not change the current value of the parameter itself.
430     */
431     public boolean canSetParameter(String name,
432                                   Object value);
433
434     /**
435     * The list of the parameters supported by this
436     * DOMConfiguration object and for which at least one value
437     * can be set by the application. Note that this list can also contain
438     * parameter names defined outside this specification.
439     */
440     public DOMStringList getParameterNames();
441
442 }

```

### 36.6 DOMError.java

Secuencia 337: [org.w3c/dom/DOMError.java](http://org.w3c/dom/DOMError.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*

```

```

26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42 package org.w3c.dom;
43
44 /**
45  * <code>DOMError</code> is an interface that describes an error.
46  * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
47  *   Object Model (DOM) Level 3 Core Specification</a>.
48  * @since DOM Level 3
49  */
49 public interface DOMError {
50     // ErrorSeverity
51     /**
52      * The severity of the error described by the <code>DOMError</code> is
53      * warning. A <code>SEVERITY_WARNING</code> will not cause the
54      * processing to stop, unless <code>DOMErrorHandler.handleError()</code>
55      * returns <code>>false</code>.
56      */
57     public static final short SEVERITY_WARNING          = 1;
58     /**
59      * The severity of the error described by the <code>DOMError</code> is
60      * error. A <code>SEVERITY_ERROR</code> may not cause the processing to
61      * stop if the error can be recovered, unless
62      * <code>DOMErrorHandler.handleError()</code> returns <code>>false</code>.
63      */
64     public static final short SEVERITY_ERROR            = 2;
65     /**
66      * The severity of the error described by the <code>DOMError</code> is
67      * fatal error. A <code>SEVERITY_FATAL_ERROR</code> will cause the
68      * normal processing to stop. The return value of
69      * <code>DOMErrorHandler.handleError()</code> is ignored unless the
70      * implementation chooses to continue, in which case the behavior
71      * becomes undefined.
72      */
73     public static final short SEVERITY_FATAL_ERROR      = 3;
74
75     /**
76      * The severity of the error, either <code>SEVERITY_WARNING</code>,
77      * <code>SEVERITY_ERROR</code>, or <code>SEVERITY_FATAL_ERROR</code>.

```



```

78     */
79     public short getSeverity();
80
81     /**
82     * An implementation specific string describing the error that occurred.
83     */
84     public String getMessage();
85
86     /**
87     * A DOMString indicating which related data is expected in
88     * relatedData. Users should refer to the specification of
89     * the error in order to find its DOMString type and
90     * relatedData definitions if any.
91     * 

<b>Note:</b> As an example,
92     * Document.normalizeDocument() does generate warnings when
93     * the "split-cdata-sections" parameter is in use. Therefore, the method
94     * generates a SEVERITY_WARNING with type
95     * "cdata-sections-split" and the first
96     * CDATASection node in document order resulting from the
97     * split is returned by the relatedData attribute.
98     */
99     public String getType();
100
101     /**
102     * The related platform dependent exception if any.
103     */
104     public Object getRelatedException();
105
106     /**
107     * The related DOMError.type dependent data if any.
108     */
109     public Object getRelatedData();
110
111     /**
112     * The location of the error.
113     */
114     public DOMLocator getLocation();
115
116 }


```

### 36.7 DOMErrorHandler.java

Secuencia 338: [org.w3c/dom/DOMErrorHandler.java](http://org.w3c/dom/DOMErrorHandler.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```



```

11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * <code>DOMErrorHandler</code> is a callback interface that the DOM
46   * implementation can call when reporting errors that happens while
47   * processing XML data, or when doing some other processing (e.g. validating
48   * a document). A <code>DOMErrorHandler</code> object can be attached to a
49   * <code>Document</code> using the "error-handler" on the
50   * <code>DOMConfiguration</code> interface. If more than one error needs to
51   * be reported during an operation, the sequence and numbers of the errors
52   * passed to the error handler are implementation dependent.
53   * <p> The application that is using the DOM implementation is expected to
54   * implement this interface.
55   * <p> See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
56   *   Object Model (DOM) Level 3 Core Specification</a>.
57   * @since DOM Level 3
58   */
59  public interface DOMErrorHandler {
60      /**
61       * This method is called on the error handler when an error occurs.
62       * <br> If an exception is thrown from this method, it is considered to be
63       * equivalent of returning <code>>true</code>.

```

```

63      * @param error The error object that describes the error. This object
64      * may be reused by the DOM implementation across multiple calls to
65      * the handleError method.
66      * @return If the handleError method returns
67      * false, the DOM implementation should stop the current
68      * processing when possible. If the method returns true,
69      * the processing may continue depending on
70      * DOMError.severity.
71      */
72      public boolean handleError(DOMError error);
73
74  }

```

### 36.8 DOMException.java

Secuencia 339: [org/w3c/dom/DOMException.java](http://org/w3c/dom/DOMException.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

```

```

38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * DOM operations only raise exceptions in "exceptional" circumstances, i.e.,
46   * when an operation is impossible to perform (either for logical reasons,
47   * because data is lost, or because the implementation has become unstable).
48   * In general, DOM methods return specific error values in ordinary
49   * processing situations, such as out-of-bound errors when using
50   * <code>NodeList</code>.
51   * 

Implementations should raise other exceptions under other circumstances.


52   * For example, implementations should raise an implementation-dependent
53   * exception if a <code>null</code> argument is passed when <code>null</code>
54   * was not expected.
55   * 

Some languages and object systems do not support the concept of
56   * exceptions. For such systems, error conditions may be indicated using
57   * native error reporting mechanisms. For some bindings, for example,
58   * methods may return error codes similar to those listed in the
59   * corresponding method descriptions.


60   * 

See also the Document
61     Object Model (DOM) Level 3 Core Specification


62  */
63  public class DOMException extends RuntimeException {
64      public DOMException(short code, String message) {
65          super(message);
66          this.code = code;
67      }
68      public short code;
69      // ExceptionCode
70      /**
71       * If index or size is negative, or greater than the allowed value.
72       */
73      public static final short INDEX_SIZE_ERR = 1;
74      /**
75       * If the specified range of text does not fit into a
76       * <code>DOMString</code>.
77       */
78      public static final short DOMSTRING_SIZE_ERR = 2;
79      /**
80       * If any <code>Node</code> is inserted somewhere it doesn't belong.
81       */
82      public static final short HIERARCHY_REQUEST_ERR = 3;
83      /**
84       * If a <code>Node</code> is used in a different document than the one
85       * that created it (that doesn't support it).
86       */
87      public static final short WRONG_DOCUMENT_ERR = 4;
88      /**
89       * If an invalid or illegal character is specified, such as in an XML name.
90       */

```

```
90 public static final short INVALID_CHARACTER_ERR = 5;
91 /**
92  * If data is specified for a <code>Node</code> which does not support
93  * data.
94  */
95 public static final short NO_DATA_ALLOWED_ERR = 6;
96 /**
97  * If an attempt is made to modify an object where modifications are not
98  * allowed.
99  */
100 public static final short NO_MODIFICATION_ALLOWED_ERR = 7;
101 /**
102  * If an attempt is made to reference a <code>Node</code> in a context
103  * where it does not exist.
104  */
105 public static final short NOT_FOUND_ERR = 8;
106 /**
107  * If the implementation does not support the requested type of object or
108  * operation.
109  */
110 public static final short NOT_SUPPORTED_ERR = 9;
111 /**
112  * If an attempt is made to add an attribute that is already in use
113  * elsewhere.
114  */
115 public static final short INUSE_ATTRIBUTE_ERR = 10;
116 /**
117  * If an attempt is made to use an object that is not, or is no longer,
118  * usable.
119  * @since DOM Level 2
120  */
121 public static final short INVALID_STATE_ERR = 11;
122 /**
123  * If an invalid or illegal string is specified.
124  * @since DOM Level 2
125  */
126 public static final short SYNTAX_ERR = 12;
127 /**
128  * If an attempt is made to modify the type of the underlying object.
129  * @since DOM Level 2
130  */
131 public static final short INVALID_MODIFICATION_ERR = 13;
132 /**
133  * If an attempt is made to create or change an object in a way which is
134  * incorrect with regard to namespaces.
135  * @since DOM Level 2
136  */
137 public static final short NAMESPACE_ERR = 14;
138 /**
139  * If a parameter or an operation is not supported by the underlying
140  * object.
141  * @since DOM Level 2
142  */
```

```

143     public static final short INVALID_ACCESS_ERR          = 15;
144     /**
145      * If a call to a method such as insertBefore or
146      * removeChild would make the Node invalid
147      * with respect to "partial validity", this exception would be raised
148      * and the operation would not be done. This code is used in [\]
150      \* . Refer to this specification for further information.
151      \* @since DOM Level 3
152      \*/
153     public static final short VALIDATION\_ERR              = 16;
154     /\*\*
155      \* If the type of an object is incompatible with the expected type of the
156      \* parameter associated to the object.
157      \* @since DOM Level 3
158      \*/
159     public static final short TYPE\_MISMATCH\_ERR          = 17;
160
161     // Added serialVersionUID to preserve binary compatibility
162     static final long serialVersionUID = 6627732366795969916L;
163 }

```

### 36.9 DOMImplementation.java

Secuencia 340: org/w3c/dom/DOMImplementation.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *

```

```

29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42 package org.w3c.dom;
43
44 /**
45  * The DOMImplementation interface provides a number of methods
46  * for performing operations that are independent of any particular instance
47  * of the document object model.
48  * 

See also the Document
49  * Object Model (DOM) Level 3 Core Specification.


50  */
51 public interface DOMImplementation {
52     /**
53      * Test if the DOM implementation implements a specific feature and
54      * version, as specified in DOM Features.
55      * @param feature The name of the feature to test.
56      * @param version This is the version number of the feature to test.
57      * @return true if the feature is implemented in the
58      * specified version, false otherwise.
59      */
60     public boolean hasFeature(String feature,
61                               String version);
62
63     /**
64      * Creates an empty DocumentType node. Entity declarations
65      * and notations are not made available. Entity reference expansions and
66      * default attribute additions do not occur..
67      * @param qualifiedName The qualified name of the document type to be
68      * created.
69      * @param publicId The external subset public identifier.
70      * @param systemId The external subset system identifier.
71      * @return A new DocumentType node with
72      * Node.ownerDocument set to null.
73      * @exception DOMException
74      *     INVALID_CHARACTER_ERR: Raised if the specified qualified name is not
75      *     an XML name according to [XML 1.0].
76      *     <br>NAMESPACE_ERR: Raised if the qualifiedName is
77      *     malformed.
78      *     <br>NOT_SUPPORTED_ERR: May be raised if the implementation does not
79      *     support the feature "XML" and the language exposed through the

```

```

79      * Document does not support XML Namespaces (such as [

```

```

126 * <br>WRONG_DOCUMENT_ERR: Raised if <code>doctype</code> has already
127 * been used with a different document or was created from a different
128 * implementation.
129 * <br>NOT_SUPPORTED_ERR: May be raised if the implementation does not
130 * support the feature "XML" and the language exposed through the
131 * Document does not support XML Namespaces (such as [in <a href="http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html#DOMFeatures">
    DOM Features</a>. The specialized object may also be obtained by using
143 * binding-specific casting methods but is not necessarily expected to,
144 * as discussed in . This method also allow the implementation to
145 * provide specialized objects which do not support the
146 * <code>DOMImplementation</code> interface.
147 * @param feature The name of the feature requested. Note that any plus
148 * sign "+" prepended to the name of the feature will be ignored since
149 * it is not significant in the context of this method.
150 * @param version This is the version number of the feature to test.
151 * @return Returns an object which implements the specialized APIs of
152 * the specified feature and version, if any, or <code>null</code> if
153 * there is no object which implements interfaces associated with that
154 * feature. If the <code>DOMObject</code> returned by this method
155 * implements the <code>DOMImplementation</code> interface, it must
156 * delegate to the primary core <code>DOMImplementation</code> and not
157 * return results inconsistent with the primary core
158 * <code>DOMImplementation</code> such as <code>hasFeature</code>,
159 * <code>getFeature</code>, etc.
160 * @since DOM Level 3
161 */
162 public Object getFeature(String feature,
163                          String version);
164
165 }

```

### 36.10 DOMImplementationList.java

Secuencia 341: org/w3c/dom/DOMImplementationList.java

```

1 /*
2 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3 *
4 *
5 *
6 *
7 *

```



```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45  * The DOMImplementationList interface provides the abstraction
46  * of an ordered collection of DOM implementations, without defining or
47  * constraining how this collection is implemented. The items in the
48  * DOMImplementationList are accessible via an integral index,
49  * starting from 0.
50  * 

See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 Document
51  * Object Model (DOM) Level 3 Core Specification.


51  * @since DOM Level 3
52  */
53 public interface DOMImplementationList {
54     /**
55      * Returns the indexth item in the collection. If
56      * index is greater than or equal to the number of
57      * DOMImplementations in the list, this returns
58      * null.
59      * @param index Index into the collection.
```

```

60     * @return The <code>DOMImplementation</code> at the <code>index</code>
61     *   th position in the <code>DOMImplementationList</code>, or
62     *   <code>null</code> if that is not a valid index.
63     */
64     public DOMImplementation item(int index);
65
66     /**
67     *   The number of <code>DOMImplementation</code>s in the list. The range
68     *   of valid child node indices is 0 to <code>length-1</code> inclusive.
69     */
70     public int getLength();
71
72 }

```

### 36.11 DOMImplementationSource.java

Secuencia 342: org/w3c/dom/DOMImplementationSource.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied

```

```

37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * This interface permits a DOM implementer to supply one or more
46   * implementations, based upon requested features and versions, as specified
47   * in in <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html#DOMFeatures'>DOM
48   * Features</a>. Each implemented DOMImplementationSource object is
49   * listed in the binding-specific list of available sources so that its
50   * DOMImplementation objects are made available.
51   * 

See also the Document
52   * Object Model (DOM) Level 3 Core Specification</a>.
53   * @since DOM Level 3
54   */
55  public interface DOMImplementationSource {
56      /**
57       * A method to request the first DOM implementation that supports the
58       * specified features.
59       * @param features A string that specifies which features and versions
60       * are required. This is a space separated list in which each feature
61       * is specified by its name optionally followed by a space and a
62       * version number. This method returns the first item of the list
63       * returned by getDOMImplementationList. As an example,
64       * the string "XML 3.0 Traversal +Events 2.0" will
65       * request a DOM implementation that supports the module "XML" for its
66       * 3.0 version, a module that support of the "Traversal" module for
67       * any version, and the module "Events" for its 2.0 version. The
68       * module "Events" must be accessible using the method
69       * Node.getFeature() and
70       * DOMImplementation.getFeature().
71       * @return The first DOM implementation that support the desired
72       * features, or null if this source has none.
73       */
74      public DOMImplementation getDOMImplementation(String features);
75
76      /**
77       * A method to request a list of DOM implementations that support the
78       * specified features and versions, as specified in .
79       * @param features A string that specifies which features and versions
80       * are required. This is a space separated list in which each feature
81       * is specified by its name optionally followed by a space and a
82       * version number. This is something like: "XML 3.0 Traversal +Events
83       * 2.0"
84       * @return A list of DOM implementations that support the desired
85       * features.
86       */
87      public DOMImplementationList getDOMImplementationList(String features);
88  }


```

**36.12** DOMLocator.java

Secuencia 343: org/w3c/dom/DOMLocator.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45 * DOMLocator is an interface that describes a location (e.g.
46 * where an error occurred).
47 * 

See also the Document
48 * Object Model (DOM) Level 3 Core Specification.
49 * @since DOM Level 3
50 */


```

```

50 public interface DOMLocator {
51     /**
52      * The line number this locator is pointing to, or <code>-1</code> if
53      * there is no column number available.
54      */
55     public int getLineNumber();
56
57     /**
58      * The column number this locator is pointing to, or <code>-1</code> if
59      * there is no column number available.
60      */
61     public int getColumnNumber();
62
63     /**
64      * The byte offset into the input source this locator is pointing to or
65      * <code>-1</code> if there is no byte offset available.
66      */
67     public int getByteOffset();
68
69     /**
70      * The UTF-16, as defined in [Unicode] and Amendment 1 of [ISO/IEC 10646], offset into the
       input source this locator is pointing to or
71      * <code>-1</code> if there is no UTF-16 offset available.
72      */
73     public int getUtf16Offset();
74
75     /**
76      * The node this locator is pointing to, or <code>null</code> if no node
77      * is available.
78      */
79     public Node getRelatedNode();
80
81     /**
82      * The URI this locator is pointing to, or <code>null</code> if no URI is
83      * available.
84      */
85     public String getUri();
86
87 }

```

### 36.13 DOMStringList.java

Secuencia 344: org/w3c/dom/DOMStringList.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```

```
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * The DOMStringList interface provides the abstraction of an
46   * ordered collection of DOMString values, without defining or
47   * constraining how this collection is implemented. The items in the
48   * DOMStringList are accessible via an integral index, starting
49   * from 0.
50   * 

See also the Document Object Model \(DOM\) Level 3 Core Specification.


51   * @since DOM Level 3
52   */
53  public interface DOMStringList {
54      /**
55       * Returns the indexth item in the collection. If
56       * index is greater than or equal to the number of
57       * DOMStrings in the list, this returns null.
58       * @param index Index into the collection.
59       * @return The DOMString at the indexth
60       * position in the DOMStringList, or null if
61       * that is not a valid index.
62       */
```

```
63     public String item(int index);
64
65     /**
66      * The number of <code>DOMString</code>s in the list. The range of valid
67      * child node indices is 0 to <code>length-1</code> inclusive.
68      */
69     public int getLength();
70
71     /**
72      * Test if a string is part of this <code>DOMStringList</code>.
73      * @param str The string to look for.
74      * @return <code>true</code> if the string has been found,
75      *         <code>false</code> otherwise.
76      */
77     public boolean contains(String str);
78
79 }
```

### 36.14 Document.java

Secuencia 345: [org.w3c/dom/Document.java](http://org.w3c/dom/Document.java)

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
```

```

33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * The Document interface represents the entire HTML or XML
46   * document. Conceptually, it is the root of the document tree, and provides
47   * the primary access to the document's data.
48   * 

Since elements, text nodes, comments, processing instructions, etc.
49   * cannot exist outside the context of a Document, the
50   * Document interface also contains the factory methods needed
51   * to create these objects. The Node objects created have a
52   * ownerDocument attribute which associates them with the
53   * Document within whose context they were created.
54   * 

See also the Document
55   * Object Model (DOM) Level 3 Core Specification


56   */
57  public interface Document extends Node {
58      /**
59       * The Document Type Declaration (see DocumentType)
60       * associated with this document. For XML documents without a document
61       * type declaration this returns null. For HTML documents,
62       * a DocumentType object may be returned, independently of
63       * the presence or absence of document type declaration in the HTML
64       * document.
65       *   
This provides direct access to the DocumentType node,
66       * child node of this Document. This node can be set at
67       * document creation time and later changed through the use of child
68       * nodes manipulation methods, such as Node.insertBefore,
69       * or Node.replaceChild. Note, however, that while some
70       * implementations may instantiate different types of
71       * Document objects supporting additional features than the
72       * "Core", such as "HTML" [DOM Level 2 HTML]
73       * , based on the DocumentType specified at creation time,
74       * changing it afterwards is very unlikely to result in a change of the
75       * features supported.
76       *
77       * @since DOM Level 3
78       */
79      public DocumentType getDoctype();
80
81      /**
82       * The DOMImplementation object that handles this document. A
83       * DOM application may use objects from multiple implementations.
84       */


```



```
84 public DOMImplementation getImplementation();
85
86 /**
87  * This is a convenience attribute that allows direct access to the child
88  * node that is the document element of the document.
89  */
90 public Element getDocumentElement();
91
92 /**
93  * Creates an element of the type specified. Note that the instance
94  * returned implements the Element interface, so attributes
95  * can be specified directly on the returned object.
96  * <br>In addition, if there are known attributes with default values,
97  * Attr nodes representing them are automatically created
98  * and attached to the element.
99  * <br>To create an element with a qualified name and namespace URI, use
100  * the createElementNS method.
101  * @param tagName The name of the element type to instantiate. For XML,
102  *   this is case-sensitive, otherwise it depends on the
103  *   case-sensitivity of the markup language in use. In that case, the
104  *   name is mapped to the canonical form of that markup by the DOM
105  *   implementation.
106  * @return A new Element object with the
107  *   nodeName attribute set to tagName, and
108  *   localName, prefix, and
109  *   namespaceURI set to null.
110  * @exception DOMException
111  *   INVALID_CHARACTER_ERR: Raised if the specified name is not an XML
112  *   name according to the XML version in use specified in the
113  *   Document.xmlVersion attribute.
114  */
115 public Element createElement(String tagName)
116     throws DOMException;
117
118 /**
119  * Creates an empty DocumentFragment object.
120  * @return A new DocumentFragment.
121  */
122 public DocumentFragment createDocumentFragment();
123
124 /**
125  * Creates a Text node given the specified string.
126  * @param data The data for the node.
127  * @return The new Text object.
128  */
129 public Text createTextNode(String data);
130
131 /**
132  * Creates a Comment node given the specified string.
133  * @param data The data for the node.
134  * @return The new Comment object.
135  */
136 public Comment createComment(String data);
```

```

137
138 /**
139  * Creates a CDATASection node whose value is the specified
140  * string.
141  * @param data The data for the CDATASection contents.
142  * @return The new CDATASection object.
143  * @exception DOMException
144  *     NOT_SUPPORTED_ERR: Raised if this document is an HTML document.
145  */
146 public CDATASection createCDATASection(String data)
147     throws DOMException;
148
149 /**
150  * Creates a ProcessingInstruction node given the specified
151  * name and data strings.
152  * @param target The target part of the processing instruction. Unlike
153  *     Document.createElementNS or
154  *     Document.createAttributeNS, no namespace well-formed
155  *     checking is done on the target name. Applications should invoke
156  *     Document.normalizeDocument() with the parameter "
157  *     namespaces" set to true in order to ensure that the
158  *     target name is namespace well-formed.
159  * @param data The data for the node.
160  * @return The new ProcessingInstruction object.
161  * @exception DOMException
162  *     INVALID_CHARACTER_ERR: Raised if the specified target is not an XML
163  *     name according to the XML version in use specified in the
164  *     Document.xmlVersion attribute.
165  *     <br>NOT_SUPPORTED_ERR: Raised if this document is an HTML document.
166  */
167 public ProcessingInstruction createProcessingInstruction(String target,
168     String data)
169     throws DOMException;
170
171 /**
172  * Creates an Attr of the given name. Note that the
173  * Attr instance can then be set on an Element
174  * using the setAttributeNode method.
175  * <br>To create an attribute with a qualified name and namespace URI, use
176  * the createAttributeNS method.
177  * @param name The name of the attribute.
178  * @return A new Attr object with the nodeName
179  *     attribute set to name, and localName,
180  *     prefix, and namespaceURI set to
181  *     null. The value of the attribute is the empty string.
182  * @exception DOMException
183  *     INVALID_CHARACTER_ERR: Raised if the specified name is not an XML
184  *     name according to the XML version in use specified in the
185  *     Document.xmlVersion attribute.
186  */
187 public Attr createAttribute(String name)
188     throws DOMException;
189

```

```

190 /**
191  * Creates an EntityReference object. In addition, if the
192  * referenced entity is known, the child list of the
193  * EntityReference node is made the same as that of the
194  * corresponding Entity node.
195  * 

><b>Note:</b> If any descendant of the Entity node has
196  * an unbound namespace prefix, the corresponding descendant of the
197  * created EntityReference node is also unbound; (its
198  * namespaceURI is null). The DOM Level 2 and
199  * 3 do not support any mechanism to resolve namespace prefixes in this
200  * case.
201  * @param name The name of the entity to reference. Unlike
202  * Document.createElementNS or
203  * Document.createAttributeNS, no namespace well-formed
204  * checking is done on the entity name. Applications should invoke
205  * Document.normalizeDocument() with the parameter "
206  * namespaces" set to true in order to ensure that the
207  * entity name is namespace well-formed.
208  * @return The new EntityReference object.
209  * @exception DOMException
210  *   INVALID_CHARACTER_ERR: Raised if the specified name is not an XML
211  *   name according to the XML version in use specified in the
212  *   Document.xmlVersion attribute.
213  *   <br>NOT_SUPPORTED_ERR: Raised if this document is an HTML document.
214  */
215 public EntityReference createEntityReference(String name)
216                                     throws DOMException;
217
218 /**
219  * Returns a NodeList of all the Elements in
220  * document order with a given tag name and are contained in the
221  * document.
222  * @param tagname The name of the tag to match on. The special value "*"
223  * matches all tags. For XML, the tagname parameter is
224  * case-sensitive, otherwise it depends on the case-sensitivity of the
225  * markup language in use.
226  * @return A new NodeList object containing all the matched
227  * Elements.
228  */
229 public NodeList getElementsByTagName(String tagname);
230
231 /**
232  * Imports a node from another document to this document, without altering
233  * or removing the source node from the original document; this method
234  * creates a new copy of the source node. The returned node has no
235  * parent; (parentNode is null).
236  * <br>For all nodes, importing a node creates a node object owned by the
237  * importing document, with attribute values identical to the source
238  * node's nodeName and nodeType, plus the
239  * attributes related to namespaces (prefix,
240  * localName, and namespaceURI). As in the
241  * cloneNode operation, the source node is not altered.
242  * User data associated to the imported node is not carried over.


```

```
243 * However, if any UserDataHandlers has been specified
244 * along with the associated data these handlers will be called with the
245 * appropriate parameters before this method returns.
246 *   
Additional information is copied as appropriate to the
247 * nodeType, attempting to mirror the behavior expected if
248 * a fragment of XML or HTML source was copied from one document to
249 * another, recognizing that the two documents may have different DTDs
250 * in the XML case. The following list describes the specifics for each
251 * type of node.
252 * 

253 * 

ATTRIBUTE_NODE

254 * :   The ownerElement attribute
    255 * is set to null and the specified flag is
    256 * set to true on the generated Attr. The
    257 * descendants of the source Attr are recursively imported
    258 * and the resulting nodes reassembled to form the corresponding subtree.
    259 * Note that the deep parameter has no effect on
    260 * Attr nodes; they always carry their children with them
    261 * when imported.

262 * 

DOCUMENT_FRAGMENT_NODE

263 * :   If the deep option
    264 * was set to true, the descendants of the source
    265 * DocumentFragment are recursively imported and the
    266 * resulting nodes reassembled under the imported
    267 * DocumentFragment to form the corresponding subtree.
    268 * Otherwise, this simply generates an empty
    269 * DocumentFragment.

270 * 

DOCUMENT_NODE

271 * :   Document
    272 * nodes cannot be imported.

273 * 

DOCUMENT_TYPE_NODE

274 * :   DocumentType
    275 * nodes cannot be imported.

276 * 

ELEMENT_NODE

277 * :   Specified attribute nodes of the source element are imported, and the generated
    278 * Attr nodes are attached to the generated
    279 * Element. Default attributes are not copied, though if the document
        being imported into defines default
    280 * attributes for this element name, those are assigned. If the
    281 * importNode deep parameter was set to
    282 * true, the descendants of the source element are
    283 * recursively imported and the resulting nodes reassembled to form the
    284 * corresponding subtree.

285 * 

ENTITY_NODE

286 * :   Entity nodes can be
    287 * imported, however in the current release of the DOM the
    288 * DocumentType is readonly. Ability to add these imported
    289 * nodes to a DocumentType will be considered for addition
    290 * to a future release of the DOM. On import, the publicId,
    291 * systemId, and notationName attributes are
    292 * copied. If a deep import is requested, the descendants
    293 * of the the source Entity are recursively imported and
    294 * the resulting nodes reassembled to form the corresponding subtree.


```

```

295 * <dt>
296 * ENTITY_REFERENCE_NODE</dt>
297 * <dd>Only the EntityReference itself is
298 * copied, even if a deep import is requested, since the
299 * source and destination documents might have defined the entity
300 * differently. If the document being imported into provides a
301 * definition for this entity name, its value is assigned.</dd>
302 * <dt>NOTATION_NODE</dt>
303 * <dd>
304 * Notation nodes can be imported, however in the current
305 * release of the DOM the DocumentType is readonly. Ability
306 * to add these imported nodes to a DocumentType will be
307 * considered for addition to a future release of the DOM. On import, the
308 * publicId and systemId attributes are copied.
309 * Note that the deep parameter has no effect on this type
310 * of nodes since they cannot have any children.</dd>
311 * <dt>
312 * PROCESSING_INSTRUCTION_NODE</dt>
313 * <dd>The imported node copies its
314 * target and data values from those of the
315 * source node. Note that the deep parameter has no effect
316 * on this type of nodes since they cannot have any children.</dd>
317 * <dt>TEXT_NODE,
318 * CDATA_SECTION_NODE, COMMENT_NODE</dt>
319 * <dd>These three types of nodes inheriting
320 * from CharacterData copy their data and
321 * length attributes from those of the source node. Note
322 * that the deep parameter has no effect on these types of
323 * nodes since they cannot have any children.</dd>
324 * </dl>
325 * @param importedNode The node to import.
326 * @param deep If true, recursively import the subtree under
327 * the specified node; if false, import only the node
328 * itself, as explained above. This has no effect on nodes that cannot
329 * have any children, and on Attr, and
330 * EntityReference nodes.
331 * @return The imported node that belongs to this Document.
332 * @exception DOMException
333 * NOT_SUPPORTED_ERR: Raised if the type of node being imported is not
334 * supported.
335 * <br>INVALID_CHARACTER_ERR: Raised if one of the imported names is not
336 * an XML name according to the XML version in use specified in the
337 * Document.xmlVersion attribute. This may happen when
338 * importing an XML 1.1 [XML 1.1]
339 * element
340 * into an XML 1.0 document, for instance.
341 * @since DOM Level 2
342 */
343 public Node importNode(Node importedNode,
344                        boolean deep)
345                        throws DOMException;
346 /**

```

```

347 * Creates an element of the given qualified name and namespace URI.
348 * <br>Per [

```

```

names-19990114/'>XML Namespaces</a>]
399 * , or if the <code>qualifiedName</code> or its prefix is "xmlns" and
400 * the <code>namespaceURI</code> is different from "<a href='http://www.w3.org/2000/xmlns/'>
  http://www.w3.org/2000/xmlns/</a>", or if the <code>namespaceURI</code> is "<a href='http
  ://www.w3.org/2000/xmlns/'>http://www.w3.org/2000/xmlns/</a>" and neither the <code>
  qualifiedName</code> nor its prefix is "xmlns".
401 * <br>NOT_SUPPORTED_ERR: Always thrown if the current document does not
402 * support the <code>"XML"</code> feature, since namespaces were
403 * defined by XML.
404 * @since DOM Level 2
405 */
406 public Element createElementNS(String namespaceURI,
407                               String qualifiedName)
408                               throws DOMException;
409
410 /**
411 * Creates an attribute of the given qualified name and namespace URI.
412 * <br>Per [<a href='http://www.w3.org/TR/1999/REC-xml-names-19990114/'>XML Namespaces</a>]
413 * , applications must use the value <code>null</code> as the
414 * <code>namespaceURI</code> parameter for methods if they wish to have
415 * no namespace.
416 * @param namespaceURI The namespace URI of the attribute to create.
417 * @param qualifiedName The qualified name of the attribute to
418 * instantiate.
419 * @return A new <code>Attr</code> object with the following attributes:
420 * <table border='1' cellpadding='3'>
421 * <tr>
422 * <th>
423 * Attribute</th>
424 * <th>Value</th>
425 * </tr>
426 * <tr>
427 * <td valign='top' rowspan='1' colspan='1'><code>Node.nodeName</code></td>
428 * <td valign='top' rowspan='1' colspan='1'>qualifiedName</td>
429 * </tr>
430 * <tr>
431 * <td valign='top' rowspan='1' colspan='1'>
432 * <code>Node.namespaceURI</code></td>
433 * <td valign='top' rowspan='1' colspan='1'><code>namespaceURI</code></td>
434 * </tr>
435 * <tr>
436 * <td valign='top' rowspan='1' colspan='1'>
437 * <code>Node.prefix</code></td>
438 * <td valign='top' rowspan='1' colspan='1'>prefix, extracted from
439 * <code>qualifiedName</code>, or <code>null</code> if there is no
440 * prefix</td>
441 * </tr>
442 * <tr>
443 * <td valign='top' rowspan='1' colspan='1'><code>Node.localName</code></td>
444 * <td valign='top' rowspan='1' colspan='1'>local name, extracted from
445 * <code>qualifiedName</code></td>
446 * </tr>
447 * <tr>

```



```

448 * <td valign='top' rowspan='1' colspan='1'><code>Attr.name</code></td>
449 * <td valign='top' rowspan='1' colspan='1'>
450 *   <code>qualifiedName</code></td>
451 * </tr>
452 * <tr>
453 * <td valign='top' rowspan='1' colspan='1'><code>Node.nodeValue</code></td>
454 * <td valign='top' rowspan='1' colspan='1'>the empty
455 *   string</td>
456 * </tr>
457 * </table>
458 * @exception DOMException
459 *   INVALID_CHARACTER_ERR: Raised if the specified
460 *   <code>qualifiedName</code> is not an XML name according to the XML
461 *   version in use specified in the <code>Document.xmlVersion</code>
462 *   attribute.
463 *   <br>NAMESPACE_ERR: Raised if the <code>qualifiedName</code> is a
464 *   malformed qualified name, if the <code>qualifiedName</code> has a
465 *   prefix and the <code>namespaceURI</code> is <code>null</code>, if
466 *   the <code>qualifiedName</code> has a prefix that is "xml" and the
467 *   <code>namespaceURI</code> is different from "<a href='http://www.w3.org/XML/1998/namespace
468 *   '>http://www.w3.org/XML/1998/namespace</a>", if the <code>qualifiedName</code> or its prefix
469 *   is "xmlns" and the
470 *   <code>namespaceURI</code> is different from "<a href='http://www.w3.org/2000/xmlns/'>http
471 *   ://www.w3.org/2000/xmlns/</a>", or if the <code>namespaceURI</code> is "<a href='http://
472 *   www.w3.org/2000/xmlns/'>http://www.w3.org/2000/xmlns/</a>" and neither the <code>
473 *   qualifiedName</code> nor its prefix is "xmlns".
474 *   <br>NOT_SUPPORTED_ERR: Always thrown if the current document does not
475 *   support the <code>"XML"</code> feature, since namespaces were
476 *   defined by XML.
477 * @since DOM Level 2
478 */
479 public Attr createAttributeNS(String namespaceURI,
480                               String qualifiedName)
481                               throws DOMException;
482
483 /**
484 * Returns a <code>NodeList</code> of all the <code>Elements</code> with a
485 * given local name and namespace URI in document order.
486 * @param namespaceURI The namespace URI of the elements to match on. The
487 *   special value <code>"*"</code> matches all namespaces.
488 * @param localName The local name of the elements to match on. The
489 *   special value "*" matches all local names.
490 * @return A new <code>NodeList</code> object containing all the matched
491 *   <code>Elements</code>.
492 * @since DOM Level 2
493 */
494 public NodeList getElementsByTagNameNS(String namespaceURI,
495                                       String localName);
496
497 /**
498 * Returns the <code>Element</code> that has an ID attribute with the
499 * given value. If no such element exists, this returns <code>null</code>

```



```

496 * . If more than one element has an ID attribute with that value, what
497 * is returned is undefined.
498 * <br> The DOM implementation is expected to use the attribute
499 * <code>Attr.isId</code> to determine if an attribute is of type ID.
500 * <p><b>Note:</b> Attributes with the name "ID" or "id" are not of type
501 * ID unless so defined.
502 * @param elementId The unique <code>id</code> value for an element.
503 * @return The matching element or <code>null</code> if there is none.
504 * @since DOM Level 2
505 */
506 public Element getElementById(String elementId);
507
508 /**
509 * An attribute specifying the encoding used for this document at the time
510 * of the parsing. This is <code>null</code> when it is not known, such
511 * as when the <code>Document</code> was created in memory.
512 * @since DOM Level 3
513 */
514 public String getInputEncoding();
515
516 /**
517 * An attribute specifying, as part of the <a href='http://www.w3.org/TR/2004/REC-xml-20040204#
    NT-XMLDecl1'>XML declaration</a>, the encoding of this document. This is <code>null</code>
    when
518 * unspecified or when it is not known, such as when the
519 * <code>Document</code> was created in memory.
520 * @since DOM Level 3
521 */
522 public String getXmlEncoding();
523
524 /**
525 * An attribute specifying, as part of the <a href='http://www.w3.org/TR/2004/REC-xml-20040204#
    NT-XMLDecl1'>XML declaration</a>, whether this document is standalone. This is <code>>false
    </code> when
526 * unspecified.
527 * <p><b>Note:</b> No verification is done on the value when setting
528 * this attribute. Applications should use
529 * <code>Document.normalizeDocument()</code> with the "validate"
530 * parameter to verify if the value matches the <a href='http://www.w3.org/TR/2004/REC-xml
    -20040204#sec-rmd'>validity
531 * constraint for standalone document declaration</a> as defined in [<a href='http://www.w3.org
    /TR/2004/REC-xml-20040204'>XML 1.0</a>].
532 * @since DOM Level 3
533 */
534 public boolean getXmlStandalone();
535
536 /**
537 * An attribute specifying, as part of the <a href='http://www.w3.org/TR/2004/REC-xml-20040204#
    NT-XMLDecl1'>XML declaration</a>, whether this document is standalone. This is <code>>false
    </code> when
538 * unspecified.
539 * <p><b>Note:</b> No verification is done on the value when setting
540 * this attribute. Applications should use
    * <code>Document.normalizeDocument()</code> with the "validate"

```

```

541     * parameter to verify if the value matches the <a href='http://www.w3.org/TR/2004/REC-xml
      -20040204#sec-rmd'>validity
542     * constraint for standalone document declaration</a> as defined in [<a href='http://www.w3.org
      /TR/2004/REC-xml-20040204'>XML 1.0</a>].
543     * @exception DOMException
544     *     NOT_SUPPORTED_ERR: Raised if this document does not support the
545     *     "XML" feature.
546     * @since DOM Level 3
547     */
548     public void setXmlStandalone(boolean xmlStandalone)
549         throws DOMException;
550
551     /**
552     * An attribute specifying, as part of the <a href='http://www.w3.org/TR/2004/REC-xml
      -20040204#NT-XMLDecl'>XML declaration</a>, the version number of this document. If there
      is no declaration and if
553     * this document supports the "XML" feature, the value is
554     * <code>"1.0"</code>. If this document does not support the "XML"
555     * feature, the value is always <code>null</code>. Changing this
556     * attribute will affect methods that check for invalid characters in
557     * XML names. Application should invoke
558     * <code>Document.normalizeDocument()</code> in order to check for
559     * invalid characters in the <code>Node</code>s that are already part of
560     * this <code>Document</code>.
561     * <br> DOM applications may use the
562     * <code>DOMImplementation.hasFeature(feature, version)</code> method
563     * with parameter values "XMLVersion" and "1.0" (respectively) to
564     * determine if an implementation supports [<a href='http://www.w3.org/TR/2004/REC-xml
      -20040204'>XML 1.0</a>]. DOM
565     * applications may use the same method with parameter values
566     * "XMLVersion" and "1.1" (respectively) to determine if an
567     * implementation supports [<a href='http://www.w3.org/TR/2004/REC-xml11-20040204/'>XML 1.1</a
      >]. In both
568     * cases, in order to support XML, an implementation must also support
569     * the "XML" feature defined in this specification. <code>Document</code>
570     * objects supporting a version of the "XMLVersion" feature must not
571     * raise a <code>NOT_SUPPORTED_ERR</code> exception for the same version
572     * number when using <code>Document.xmlVersion</code>.
573     * @since DOM Level 3
574     */
575     public String getXmlVersion();
576
577     /**
578     * An attribute specifying, as part of the <a href='http://www.w3.org/TR/2004/REC-xml
      -20040204#NT-XMLDecl'>XML declaration</a>, the version number of this document. If there
      is no declaration and if
579     * this document supports the "XML" feature, the value is
580     * <code>"1.0"</code>. If this document does not support the "XML"
581     * feature, the value is always <code>null</code>. Changing this
582     * attribute will affect methods that check for invalid characters in
583     * XML names. Application should invoke
584     * <code>Document.normalizeDocument()</code> in order to check for
585     * invalid characters in the <code>Node</code>s that are already part of
586     * this <code>Document</code>.

```

```

586 * <br> DOM applications may use the
587 * <code>DOMImplementation.hasFeature(feature, version)</code> method
588 * with parameter values "XMLVersion" and "1.0" (respectively) to
589 * determine if an implementation supports [

```

```

637 * <br> Beware that when the <code>Document</code> supports the feature
638 * "HTML" [<a href='http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109'>DOM Level 2 HTML</a>]
639 * , the href attribute of the HTML BASE element takes precedence over
640 * this attribute when computing <code>Node.baseURI</code>.
641 * @since DOM Level 3
642 */
643 public String getDocumentURI();
644 /**
645 * The location of the document or <code>null</code> if undefined or if
646 * the <code>Document</code> was created using
647 * <code>DOMImplementation.createDocument</code>. No lexical checking is
648 * performed when setting this attribute; this could result in a
649 * <code>null</code> value returned when using <code>Node.baseURI</code>
650 * .
651 * <br> Beware that when the <code>Document</code> supports the feature
652 * "HTML" [<a href='http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109'>DOM Level 2 HTML</a>]
653 * , the href attribute of the HTML BASE element takes precedence over
654 * this attribute when computing <code>Node.baseURI</code>.
655 * @since DOM Level 3
656 */
657 public void setDocumentURI(String documentURI);
658
659 /**
660 * Attempts to adopt a node from another document to this document. If
661 * supported, it changes the <code>ownerDocument</code> of the source
662 * node, its children, as well as the attached attribute nodes if there
663 * are any. If the source node has a parent it is first removed from the
664 * child list of its parent. This effectively allows moving a subtree
665 * from one document to another (unlike <code>importNode()</code> which
666 * create a copy of the source node instead of moving it). When it
667 * fails, applications should use <code>Document.importNode()</code>
668 * instead. Note that if the adopted node is already part of this
669 * document (i.e. the source and target document are the same), this
670 * method still has the effect of removing the source node from the
671 * child list of its parent, if any. The following list describes the
672 * specifics for each type of node.
673 * <dl>
674 * <dt>ATTRIBUTE_NODE</dt>
675 * <dd>The
676 * <code>ownerElement</code> attribute is set to <code>null</code> and
677 * the <code>specified</code> flag is set to <code>true</code> on the
678 * adopted <code>Attr</code>. The descendants of the source
679 * <code>Attr</code> are recursively adopted.</dd>
680 * <dt>DOCUMENT_FRAGMENT_NODE</dt>
681 * <dd>The
682 * descendants of the source node are recursively adopted.</dd>
683 * <dt>DOCUMENT_NODE</dt>
684 * <dd>
685 * <code>Document</code> nodes cannot be adopted.</dd>
686 * <dt>DOCUMENT_TYPE_NODE</dt>
687 * <dd>

```

```

688 * <code>DocumentType</code> nodes cannot be adopted.</dd>
689 * <dt>ELEMENT_NODE</dt>
690 * <dd><em>Specified</em> attribute nodes of the source element are adopted. Default attributes
691 * are discarded, though if the document being adopted into defines
692 * default attributes for this element name, those are assigned. The
693 * descendants of the source element are recursively adopted.</dd>
694 * <dt>ENTITY_NODE</dt>
695 * <dd>
696 * <code>Entity</code> nodes cannot be adopted.</dd>
697 * <dt>ENTITY_REFERENCE_NODE</dt>
698 * <dd>Only
699 * the <code>EntityReference</code> node itself is adopted, the
700 * descendants are discarded, since the source and destination documents
701 * might have defined the entity differently. If the document being
702 * imported into provides a definition for this entity name, its value
703 * is assigned.</dd>
704 * <dt>NOTATION_NODE</dt>
705 * <dd><code>Notation</code> nodes cannot be
706 * adopted.</dd>
707 * <dt>PROCESSING_INSTRUCTION_NODE, TEXT_NODE, CDATA_SECTION_NODE,
708 * COMMENT_NODE</dt>
709 * <dd>These nodes can all be adopted. No specifics.</dd>
710 * </dl>
711 * <p><b>Note:</b> Since it does not create new nodes unlike the
712 * <code>Document.importNode()</code> method, this method does not raise
713 * an <code>INVALID_CHARACTER_ERR</code> exception, and applications
714 * should use the <code>Document.normalizeDocument()</code> method to
715 * check if an imported name is not an XML name according to the XML
716 * version in use.
717 * @param source The node to move into this document.
718 * @return The adopted node, or <code>null</code> if this operation
719 * fails, such as when the source node comes from a different
720 * implementation.
721 * @exception DOMException
722 * NOT_SUPPORTED_ERR: Raised if the source node is of type
723 * <code>DOCUMENT</code>, <code>DOCUMENT_TYPE</code>.
724 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised when the source node is
725 * readonly.
726 * @since DOM Level 3
727 */
728 public Node adoptNode(Node source)
729     throws DOMException;
730
731 /**
732 * The configuration used when <code>Document.normalizeDocument()</code>
733 * is invoked.
734 * @since DOM Level 3
735 */
736 public DOMConfiguration getDomConfig();
737
738 /**
739 * This method acts as if the document was going through a save and load
740 * cycle, putting the document in a "normal" form. As a consequence,

```

```

741 * this method updates the replacement tree of
742 * <code>EntityReference</code> nodes and normalizes <code>Text</code>
743 * nodes, as defined in the method <code>Node.normalize()</code>.
744 * <br> Otherwise, the actual result depends on the features being set on
745 * the <code>Document.domConfig</code> object and governing what
746 * operations actually take place. Noticeably this method could also
747 * make the document namespace well-formed according to the algorithm
748 * described in , check the character normalization, remove the
749 * <code>CDATASection</code> nodes, etc. See
750 * <code>DOMConfiguration</code> for details.
751 * <pre>// Keep in the document
752 * the information defined // in the XML Information Set (Java example)
753 * DOMConfiguration docConfig = myDocument.getDomConfig();
754 * docConfig.setParameter("infoset", Boolean.TRUE);
755 * myDocument.normalizeDocument();</pre>
756 *
757 * <br>Mutation events, when supported, are generated to reflect the
758 * changes occurring on the document.
759 * <br> If errors occur during the invocation of this method, such as an
760 * attempt to update a read-only node or a <code>Node.nodeName</code>
761 * contains an invalid character according to the XML version in use,
762 * errors or warnings (<code>DOMError.SEVERITY_ERROR</code> or
763 * <code>DOMError.SEVERITY_WARNING</code>) will be reported using the
764 * <code>DOMErrorHandler</code> object associated with the "error-handler
765 * " parameter. Note this method might also report fatal errors (
766 * <code>DOMError.SEVERITY_FATAL_ERROR</code>) if an implementation
767 * cannot recover from an error.
768 * @since DOM Level 3
769 */
770 public void normalizeDocument();
771
772 /**
773 * Rename an existing node of type <code>ELEMENT_NODE</code> or
774 * <code>ATTRIBUTE_NODE</code>.
775 * <br>When possible this simply changes the name of the given node,
776 * otherwise this creates a new node with the specified name and
777 * replaces the existing node with the new node as described below.
778 * <br>If simply changing the name of the given node is not possible, the
779 * following operations are performed: a new node is created, any
780 * registered event listener is registered on the new node, any user
781 * data attached to the old node is removed from that node, the old node
782 * is removed from its parent if it has one, the children are moved to
783 * the new node, if the renamed node is an <code>Element</code> its
784 * attributes are moved to the new node, the new node is inserted at the
785 * position the old node used to have in its parent's child nodes list
786 * if it has one, the user data that was attached to the old node is
787 * attached to the new node.
788 * <br>When the node being renamed is an <code>Element</code> only the
789 * specified attributes are moved, default attributes originated from
790 * the DTD are updated according to the new element name. In addition,
791 * the implementation may update default attributes from other schemas.
792 * Applications should use <code>Document.normalizeDocument()</code> to
793 * guarantee these attributes are up-to-date.

```



```

794 * <br>When the node being renamed is an <code>Attr</code> that is
795 * attached to an <code>Element</code>, the node is first removed from
796 * the <code>Element</code> attributes map. Then, once renamed, either
797 * by modifying the existing node or creating a new one as described
798 * above, it is put back.
799 * <br>In addition,
800 * <ul>
801 * <li> a user data event <code>NODE_RENAMED</code> is fired,
802 * </li>
803 * <li>
804 * when the implementation supports the feature "MutationNameEvents",
805 * each mutation operation involved in this method fires the appropriate
806 * event, and in the end the event {
807 * <code>http://www.w3.org/2001/xml-events</code>,
808 * <code>DOMElementNameChanged</code>} or {
809 * <code>http://www.w3.org/2001/xml-events</code>,
810 * <code>DOMAttributeNameChanged</code>} is fired.
811 * </li>
812 * </ul>
813 * @param n The node to rename.
814 * @param namespaceURI The new namespace URI.
815 * @param qualifiedName The new qualified name.
816 * @return The renamed node. This is either the specified node or the new
817 * node that was created to replace the specified node.
818 * @exception DOMException
819 * NOT_SUPPORTED_ERR: Raised when the type of the specified node is
820 * neither <code>ELEMENT_NODE</code> nor <code>ATTRIBUTE_NODE</code>,
821 * or if the implementation does not support the renaming of the
822 * document element.
823 * <br>INVALID_CHARACTER_ERR: Raised if the new qualified name is not an
824 * XML name according to the XML version in use specified in the
825 * <code>Document.xmlVersion</code> attribute.
826 * <br>WRONG_DOCUMENT_ERR: Raised when the specified node was created
827 * from a different document than this document.
828 * <br>NAMESPACE_ERR: Raised if the <code>qualifiedName</code> is a
829 * malformed qualified name, if the <code>qualifiedName</code> has a
830 * prefix and the <code>namespaceURI</code> is <code>null</code>, or
831 * if the <code>qualifiedName</code> has a prefix that is "xml" and
832 * the <code>namespaceURI</code> is different from "<a href='http://www.w3.org/XML/1998/
833 * namespace'>
834 * http://www.w3.org/XML/1998/namespace</a>" [<a href='http://www.w3.org/TR/1999/REC-xml-
835 * names-19990114/'>XML Namespaces</a>]
836 * . Also raised, when the node being renamed is an attribute, if the
837 * <code>qualifiedName</code>, or its prefix, is "xmlns" and the
838 * <code>namespaceURI</code> is different from "<a href='http://www.w3.org/2000/xmlns/'>http
839 * ://www.w3.org/2000/xmlns/</a>".
840 * @since DOM Level 3
841 */
842 public Node renameNode(Node n,
843     String namespaceURI,
844     String qualifiedName)
845     throws DOMException;

```

844 }

**36.15 DocumentFragment.java**

Secuencia 346: org/w3c/dom/DocumentFragment.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45 * <code>DocumentFragment</code> is a "lightweight" or "minimal"
46 * <code>Document</code> object. It is very common to want to be able to
47 * extract a portion of a document's tree or to create a new fragment of a
48 * document. Imagine implementing a user command like cut or rearranging a
```



```

49  * document by moving fragments around. It is desirable to have an object
50  * which can hold such fragments and it is quite natural to use a Node for
51  * this purpose. While it is true that a <code>Document</code> object could
52  * fulfill this role, a <code>Document</code> object can potentially be a
53  * heavyweight object, depending on the underlying implementation. What is
54  * really needed for this is a very lightweight object.
55  * <code>DocumentFragment</code> is such an object.
56  * <p>Furthermore, various operations -- such as inserting nodes as children
57  * of another <code>Node</code> -- may take <code>DocumentFragment</code>
58  * objects as arguments; this results in all the child nodes of the
59  * <code>DocumentFragment</code> being moved to the child list of this node.
60  * <p>The children of a <code>DocumentFragment</code> node are zero or more
61  * nodes representing the tops of any sub-trees defining the structure of
62  * the document. <code>DocumentFragment</code> nodes do not need to be
63  * well-formed XML documents (although they do need to follow the rules
64  * imposed upon well-formed XML parsed entities, which can have multiple top
65  * nodes). For example, a <code>DocumentFragment</code> might have only one
66  * child and that child node could be a <code>Text</code> node. Such a
67  * structure model represents neither an HTML document nor a well-formed XML
68  * document.
69  * <p>When a <code>DocumentFragment</code> is inserted into a
70  * <code>Document</code> (or indeed any other <code>Node</code> that may
71  * take children) the children of the <code>DocumentFragment</code> and not
72  * the <code>DocumentFragment</code> itself are inserted into the
73  * <code>Node</code>. This makes the <code>DocumentFragment</code> very
74  * useful when the user wishes to create nodes that are siblings; the
75  * <code>DocumentFragment</code> acts as the parent of these nodes so that
76  * the user can use the standard methods from the <code>Node</code>
77  * interface, such as <code>Node.insertBefore</code> and
78  * <code>Node.appendChild</code>.
79  * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
80  */
81  public interface DocumentFragment extends Node {
82  }

```

### 36.16 DocumentType.java

Secuencia 347: org/w3c/dom/DocumentType.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```

```

15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * Each Document has a doctype attribute whose value
46   * is either null or a DocumentType object. The
47   * DocumentType interface in the DOM Core provides an interface
48   * to the list of entities that are defined for the document, and little
49   * else because the effect of namespaces and the various XML schema efforts
50   * on DTD representation are not clearly understood as of this writing.
51   * 

DOM Level 3 doesn't support editing DocumentType nodes.


52   * DocumentType nodes are read-only.
53   * 

See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 Document
54   * Object Model (DOM) Level 3 Core Specification


55   */
56  public interface DocumentType extends Node {
57      /**
58       * The name of DTD; i.e., the name immediately following the
59       * DOCTYPE keyword.
60       */
61      public String getName();
62
63      /**
64       * A NamedNodeMap containing the general entities, both
65       * external and internal, declared in the DTD. Parameter entities are
66       * not contained. Duplicates are discarded. For example in:
67       * 

```
<pre><!DOCTYPE
```


```

```

67      * ex SYSTEM "ex.dtd" [ &lt;!ENTITY foo "foo"&gt; &lt;!ENTITY bar
68      * "bar"&gt; &lt;!ENTITY bar "bar2"&gt; &lt;!ENTITY % baz "baz"&gt;
69      * ]&gt; &lt;ex/&gt;</pre>
70      * the interface provides access to <code>foo</code>
71      * and the first declaration of <code>bar</code> but not the second
72      * declaration of <code>bar</code> or <code>baz</code>. Every node in
73      * this map also implements the <code>Entity</code> interface.
74      * <br>The DOM Level 2 does not support editing entities, therefore
75      * <code>entities</code> cannot be altered in any way.
76      */
77      public NamedNodeMap getEntities();
78
79      /**
80      * A <code>NamedNodeMap</code> containing the notations declared in the
81      * DTD. Duplicates are discarded. Every node in this map also implements
82      * the <code>Notation</code> interface.
83      * <br>The DOM Level 2 does not support editing notations, therefore
84      * <code>notations</code> cannot be altered in any way.
85      */
86      public NamedNodeMap getNotations();
87
88      /**
89      * The public identifier of the external subset.
90      * @since DOM Level 2
91      */
92      public String getPublicId();
93
94      /**
95      * The system identifier of the external subset. This may be an absolute
96      * URI or not.
97      * @since DOM Level 2
98      */
99      public String getSystemId();
100
101      /**
102      * The internal subset as a string, or <code>null</code> if there is none.
103      * This is does not contain the delimiting square brackets.
104      * <p><b>Note:</b> The actual content returned depends on how much
105      * information is available to the implementation. This may vary
106      * depending on various parameters, including the XML processor used to
107      * build the document.
108      * @since DOM Level 2
109      */
110      public String getInternalSubset();
111
112 }

```

### 36.17 Element.java

Secuencia 348: org/w3c/dom/Element.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45  * The Element interface represents an element in an HTML or XML
46  * document. Elements may have attributes associated with them; since the
47  * Element interface inherits from Node, the
48  * generic Node interface attribute attributes may
49  * be used to retrieve the set of all attributes for an element. There are
50  * methods on the Element interface to retrieve either an
51  * Attr object by name or an attribute value by name. In XML,
52  * where an attribute value may contain entity references, an
53  * Attr object should be retrieved to examine the possibly
54  * fairly complex sub-tree representing the attribute value. On the other
55  * hand, in HTML, where all attributes have simple string values, methods to
56  * directly access an attribute value can safely be used as a convenience.
```

```

57 * <p><b>Note:</b> In DOM Level 2, the method normalize is
58 * inherited from the Node interface where it was moved.
59 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
60 */
61 public interface Element extends Node {
62     /**
63      * The name of the element. If Node.localName is different
64      * from null, this attribute is a qualified name. For
65      * example, in:
66      * <pre> <elementExample id="demo"> ...
67      * </elementExample> , </pre>
68      * tagName has the value
69      * "elementExample". Note that this is case-preserving in
70      * XML, as are all of the operations of the DOM. The HTML DOM returns
71      * the tagName of an HTML element in the canonical
72      * uppercase form, regardless of the case in the source HTML document.
73     */
74     public String getTagName();
75
76     /**
77      * Retrieves an attribute value by name.
78      * @param name The name of the attribute to retrieve.
79      * @return The Attr value as a string, or the empty string
80      *         if that attribute does not have a specified or default value.
81     */
82     public String getAttribute(String name);
83
84     /**
85      * Adds a new attribute. If an attribute with that name is already present
86      * in the element, its value is changed to be that of the value
87      * parameter. This value is a simple string; it is not parsed as it is
88      * being set. So any markup (such as syntax to be recognized as an
89      * entity reference) is treated as literal text, and needs to be
90      * appropriately escaped by the implementation when it is written out.
91      * In order to assign an attribute value that contains entity
92      * references, the user must create an Attr node plus any
93      * Text and EntityReference nodes, build the
94      * appropriate subtree, and use setAttributeNode to assign
95      * it as the value of an attribute.
96      * <br>To set an attribute with a qualified name and namespace URI, use
97      * the setAttributeNS method.
98      * @param name The name of the attribute to create or alter.
99      * @param value Value to set in string form.
100     * @exception DOMException
101     *     INVALID_CHARACTER_ERR: Raised if the specified name is not an XML
102     *     name according to the XML version in use specified in the
103     *     Document.xmlVersion attribute.
104     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
105     */
106     public void setAttribute(String name,
107                             String value)
108                             throws DOMException;

```

```

109
110 /**
111  * Removes an attribute by name. If a default value for the removed
112  * attribute is defined in the DTD, a new attribute immediately appears
113  * with the default value as well as the corresponding namespace URI,
114  * local name, and prefix when applicable. The implementation may handle
115  * default values from other schemas similarly but applications should
116  * use Document.normalizeDocument() to guarantee this
117  * information is up-to-date.
118  *   
If no attribute with this name is found, this method has no effect.
119  *   
To remove an attribute by local name and namespace URI, use the
120  * removeAttributeNS method.
121  * @param name The name of the attribute to remove.
122  * @exception DOMException
123  *   NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
124  */
125 public void removeAttribute(String name)
126     throws DOMException;
127
128 /**
129  * Retrieves an attribute node by name.
130  *   
To retrieve an attribute node by qualified name and namespace URI,
131  * use the getAttributeNodeNS method.
132  * @param name The name (nodeName) of the attribute to
133  *   retrieve.
134  * @return The Attr node with the specified name (
135  *   nodeName) or null if there is no such
136  *   attribute.
137  */
138 public Attr getAttributeNode(String name);
139
140 /**
141  * Adds a new attribute node. If an attribute with that name (
142  * nodeName) is already present in the element, it is
143  * replaced by the new one. Replacing an attribute node by itself has no
144  * effect.
145  *   
To add a new attribute node with a qualified name and namespace
146  * URI, use the setAttributeNodeNS method.
147  * @param newAttr The Attr node to add to the attribute list.
148  * @return If the newAttr attribute replaces an existing
149  *   attribute, the replaced Attr node is returned,
150  *   otherwise null is returned.
151  * @exception DOMException
152  *   WRONG_DOCUMENT_ERR: Raised if newAttr was created from a
153  *   different document than the one that created the element.
154  *     
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
155  *     
INUSE_ATTRIBUTE_ERR: Raised if newAttr is already an
156  *   attribute of another Element object. The DOM user must
157  *   explicitly clone Attr nodes to re-use them in other
158  *   elements.
159  */
160 public Attr setAttributeNode(Attr newAttr)
161     throws DOMException;

```

```

162
163 /**
164  * Removes the specified attribute node. If a default value for the
165  * removed Attr node is defined in the DTD, a new node
166  * immediately appears with the default value as well as the
167  * corresponding namespace URI, local name, and prefix when applicable.
168  * The implementation may handle default values from other schemas
169  * similarly but applications should use
170  * Document.normalizeDocument() to guarantee this
171  * information is up-to-date.
172  * @param oldAttr The Attr node to remove from the attribute
173  *   list.
174  * @return The Attr node that was removed.
175  * @exception DOMException
176  *   NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
177  *   NOT_FOUND_ERR: Raised if oldAttr is not an attribute
178  *   of the element.
179  */
180 public Attr removeAttributeNode(Attr oldAttr)
181     throws DOMException;
182
183 /**
184  * Returns a NodeList of all descendant Elements
185  * with a given tag name, in document order.
186  * @param name The name of the tag to match on. The special value "*"
187  *   matches all tags.
188  * @return A list of matching Element nodes.
189  */
190 public NodeList getElementsByTagName(String name);
191
192 /**
193  * Retrieves an attribute value by local name and namespace URI.
194  *   
Per [null as the
196  \* namespaceURI parameter for methods if they wish to have
197  \* no namespace.
198  \* @param namespaceURI The namespace URI of the attribute to retrieve.
199  \* @param localName The local name of the attribute to retrieve.
200  \* @return The Attr value as a string, or the empty string
201  \*   if that attribute does not have a specified or default value.
202  \* @exception DOMException
203  \*   NOT\_SUPPORTED\_ERR: May be raised if the implementation does not
204  \*   support the feature "XML" and the language exposed
205  \*   through the Document does not support XML Namespaces \(such as \[

```

```

214 * namespace URI is already present on the element, its prefix is
215 * changed to be the prefix part of the <code>qualifiedName</code>, and
216 * its value is changed to be the <code>value</code> parameter. This
217 * value is a simple string; it is not parsed as it is being set. So any
218 * markup (such as syntax to be recognized as an entity reference) is
219 * treated as literal text, and needs to be appropriately escaped by the
220 * implementation when it is written out. In order to assign an
221 * attribute value that contains entity references, the user must create
222 * an <code>Attr</code> node plus any <code>Text</code> and
223 * <code>EntityReference</code> nodes, build the appropriate subtree,
224 * and use <code>setAttributeNodeNS</code> or
225 * <code>setAttributeNode</code> to assign it as the value of an
226 * attribute.
227 * <br>Per [

```



```

261 * value for the removed attribute is defined in the DTD, a new
262 * attribute immediately appears with the default value as well as the
263 * corresponding namespace URI, local name, and prefix when applicable.
264 * The implementation may handle default values from other schemas
265 * similarly but applications should use
266 * Document.normalizeDocument() to guarantee this
267 * information is up-to-date.
268 *   
If no attribute with this local name and namespace URI is found,
269 * this method has no effect.
270 *   
Per [null as the
272 \* namespaceURI parameter for methods if they wish to have
273 \* no namespace.
274 \* @param namespaceURI The namespace URI of the attribute to remove.
275 \* @param localName The local name of the attribute to remove.
276 \* @exception DOMException
277 \* NO\_MODIFICATION\_ALLOWED\_ERR: Raised if this node is readonly.
278 \*   
NOT\_SUPPORTED\_ERR: May be raised if the implementation does not
279 \* support the feature "XML" and the language exposed
280 \* through the Document does not support XML Namespaces \(such as \[Attr node by local name and namespace URI.
289 \\*   
Per \\[null as the
291 \\\* namespaceURI parameter for methods if they wish to have
292 \\\* no namespace.
293 \\\* @param namespaceURI The namespace URI of the attribute to retrieve.
294 \\\* @param localName The local name of the attribute to retrieve.
295 \\\* @return The Attr node with the specified attribute local
296 \\\* name and namespace URI or null if there is no such
297 \\\* attribute.
298 \\\* @exception DOMException
299 \\\* NOT\\\_SUPPORTED\\\_ERR: May be raised if the implementation does not
300 \\\* support the feature "XML" and the language exposed
301 \\\* through the Document does not support XML Namespaces \\\(such as \\\[

```

```

312 * <br>Per [

```

```

363     * specified on this element or has a default value, false
364     * otherwise.
365     * @since DOM Level 2
366     */
367     public boolean hasAttribute(String name);
368
369     /**
370     * Returns true when an attribute with a given local name and
371     * namespace URI is specified on this element or has a default value,
372     * false otherwise.
373     *   
Per [\]
374     \* , applications must use the value null as the
375     \* namespaceURI parameter for methods if they wish to have
376     \* no namespace.
377     \* @param namespaceURI The namespace URI of the attribute to look for.
378     \* @param localName The local name of the attribute to look for.
379     \* @return true if an attribute with the given local name
380     \* and namespace URI is specified or has a default value on this
381     \* element, false otherwise.
382     \* @exception DOMException
383     \* NOT\_SUPPORTED\_ERR: May be raised if the implementation does not
384     \* support the feature "XML" and the language exposed
385     \* through the Document does not support XML Namespaces \(such as \[\\]\\).
386     \\* @since DOM Level 2
387     \\*/
388     public boolean hasAttributeNS\\(String namespaceURI,
389                                   String localName\\)
390                                   throws DOMException;
391
392     /\\*\\*
393     \\* The type information associated with this element.
394     \\* @since DOM Level 3
395     \\*/
396     public TypeInfo getSchemaTypeInfo\\(\\);
397
398     /\\*\\*
399     \\* If the parameter isId is true, this method
400     \\* declares the specified attribute to be a user-determined ID attribute
401     \\* . This affects the value of Attr.isId and the behavior
402     \\* of Document.getElementById, but does not change any
403     \\* schema that may be in use, in particular this does not affect the
404     \\* Attr.schemaTypeInfo of the specified Attr
405     \\* node. Use the value false for the parameter
406     \\* isId to undeclare an attribute for being a
407     \\* user-determined ID attribute.
408     \\*   
To specify an attribute by local name and namespace URI, use the
409     \\* setIdAttributeNS method.
410     \\* @param name The name of the attribute.
411     \\* @param isId Whether the attribute is a of type ID.
412     \\* @exception DOMException
413     \\* NO\\_MODIFICATION\\_ALLOWED\\_ERR: Raised if this node is readonly.
414     \\*   
NOT\\_FOUND\\_ERR: Raised if the specified node is not an attribute

```



468 }

**36.18 Entity.java**

Secuencia 349: org/w3c/dom/Entity.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45 * This interface represents a known entity, either parsed or unparsed, in an
46 * XML document. Note that this models the entity itself <em>not</em> the entity declaration.
47 * <p>The <code>nodeName</code> attribute that is inherited from
48 * <code>Node</code> contains the name of the entity.
```

```

49 * <p>An XML processor may choose to completely expand entities before the
50 * structure model is passed to the DOM; in this case there will be no
51 * <code>EntityReference</code> nodes in the document tree.
52 * <p>XML does not mandate that a non-validating XML processor read and
53 * process entity declarations made in the external subset or declared in
54 * parameter entities. This means that parsed entities declared in the
55 * external subset need not be expanded by some classes of applications, and
56 * that the replacement text of the entity may not be available. When the <a href='http://www.w3.
    org/TR/2004/REC-xml-20040204#intern-replacement'>
57 * replacement text</a> is available, the corresponding <code>Entity</code> node's child list
58 * represents the structure of that replacement value. Otherwise, the child
59 * list is empty.
60 * <p>DOM Level 3 does not support editing <code>Entity</code> nodes; if a
61 * user wants to make changes to the contents of an <code>Entity</code>,
62 * every related <code>EntityReference</code> node has to be replaced in the
63 * structure model by a clone of the <code>Entity</code>'s contents, and
64 * then the desired changes must be made to each of those clones instead.
65 * <code>Entity</code> nodes and all their descendants are readonly.
66 * <p>An <code>Entity</code> node does not have any parent.
67 * <p><b>Note:</b> If the entity contains an unbound namespace prefix, the
68 * <code>namespaceURI</code> of the corresponding node in the
69 * <code>Entity</code> node subtree is <code>null</code>. The same is true
70 * for <code>EntityReference</code> nodes that refer to this entity, when
71 * they are created using the <code>createEntityReference</code> method of
72 * the <code>Document</code> interface.
73 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
74 */
75 public interface Entity extends Node {
76     /**
77      * The public identifier associated with the entity if specified, and
78      * <code>null</code> otherwise.
79      */
80     public String getPublicId();
81
82     /**
83      * The system identifier associated with the entity if specified, and
84      * <code>null</code> otherwise. This may be an absolute URI or not.
85      */
86     public String getSystemId();
87
88     /**
89      * For unparsed entities, the name of the notation for the entity. For
90      * parsed entities, this is <code>null</code>.
91      */
92     public String getNotationName();
93
94     /**
95      * An attribute specifying the encoding used for this entity at the time
96      * of parsing, when it is an external parsed entity. This is
97      * <code>null</code> if it an entity from the internal subset or if it
98      * is not known.
99      * @since DOM Level 3

```

```
100     */
101     public String getInputEncoding();
102
103     /**
104      * An attribute specifying, as part of the text declaration, the encoding
105      * of this entity, when it is an external parsed entity. This is
106      * <code>null</code> otherwise.
107      * @since DOM Level 3
108      */
109     public String getXmlEncoding();
110
111     /**
112      * An attribute specifying, as part of the text declaration, the version
113      * number of this entity, when it is an external parsed entity. This is
114      * <code>null</code> otherwise.
115      * @since DOM Level 3
116      */
117     public String getXmlVersion();
118
119 }
```

### 36.19 EntityReference.java

Secuencia 350: org/w3c/dom/EntityReference.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
```

```

30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42 package org.w3c.dom;
43
44 /**
45  * <code>EntityReference</code> nodes may be used to represent an entity
46  * reference in the tree. Note that character references and references to
47  * predefined entities are considered to be expanded by the HTML or XML
48  * processor so that characters are represented by their Unicode equivalent
49  * rather than by an entity reference. Moreover, the XML processor may
50  * completely expand references to entities while building the
51  * <code>Document</code>, instead of providing <code>EntityReference</code>
52  * nodes. If it does provide such nodes, then for an
53  * <code>EntityReference</code> node that represents a reference to a known
54  * entity an <code>Entity</code> exists, and the subtree of the
55  * <code>EntityReference</code> node is a copy of the <code>Entity</code>
56  * node subtree. However, the latter may not be true when an entity contains
57  * an unbound namespace prefix. In such a case, because the namespace prefix
58  * resolution depends on where the entity reference is, the descendants of
59  * the <code>EntityReference</code> node may be bound to different namespace
60  * URIs. When an <code>EntityReference</code> node represents a reference to
61  * an unknown entity, the node has no children and its replacement value,
62  * when used by <code>Attr.value</code> for example, is empty.
63  * <p>As for <code>Entity</code> nodes, <code>EntityReference</code> nodes and
64  * all their descendants are readonly.
65  * <p><b>Note:</b> <code>EntityReference</code> nodes may cause element
66  * content and attribute value normalization problems when, such as in XML
67  * 1.0 and XML Schema, the normalization is performed after entity reference
68  * are expanded.
69  * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
70  */
71 public interface EntityReference extends Node {
72 }

```

## 36.20 NameList.java

Secuencia 351: org/w3c/dom/NameList.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *

```



```

6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45  * The NameList interface provides the abstraction of an ordered
46  * collection of parallel pairs of name and namespace values (which could be
47  * null values), without defining or constraining how this collection is
48  * implemented. The items in the NameList are accessible via an
49  * integral index, starting from 0.
50  * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
51  * @since DOM Level 3
52  */
53 public interface NameList {
54     /**
55      * Returns the indexth name item in the collection.
56      * @param index Index into the collection.
57      * @return The name at the indexth position in the

```

```

58     * <code>NameList</code>, or <code>null</code> if there is no name for
59     * the specified index or if the index is out of range.
60     */
61     public String getName(int index);
62
63     /**
64     * Returns the <code>index</code>th namespaceURI item in the collection.
65     * @param index Index into the collection.
66     * @return The namespace URI at the <code>index</code>th position in the
67     * <code>NameList</code>, or <code>null</code> if there is no name for
68     * the specified index or if the index is out of range.
69     */
70     public String getNamespaceURI(int index);
71
72     /**
73     * The number of pairs (name and namespaceURI) in the list. The range of
74     * valid child node indices is 0 to <code>length-1</code> inclusive.
75     */
76     public int getLength();
77
78     /**
79     * Test if a name is part of this <code>NameList</code>.
80     * @param str The name to look for.
81     * @return <code>true</code> if the name has been found,
82     * <code>false</code> otherwise.
83     */
84     public boolean contains(String str);
85
86     /**
87     * Test if the pair namespaceURI/name is part of this
88     * <code>NameList</code>.
89     * @param namespaceURI The namespace URI to look for.
90     * @param name The name to look for.
91     * @return <code>true</code> if the pair namespaceURI/name has been
92     * found, <code>false</code> otherwise.
93     */
94     public boolean containsNS(String namespaceURI,
95                             String name);
96
97 }

```

### 36.21 NamedNodeMap.java

Secuencia 352: org/w3c/dom/NamedNodeMap.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *

```

```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * Objects implementing the NamedNodeMap interface are used to
46   * represent collections of nodes that can be accessed by name. Note that
47   * NamedNodeMap does not inherit from NodeList;
48   * NamedNodeMaps are not maintained in any particular order.
49   * Objects contained in an object implementing NamedNodeMap may
50   * also be accessed by an ordinal index, but this is simply to allow
51   * convenient enumeration of the contents of a NamedNodeMap,
52   * and does not imply that the DOM specifies an order to these Nodes.
53   * 

NamedNodeMap objects in the DOM are live.


54   * 

See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 Document
55   * Object Model (DOM) Level 3 Core Specification


56   */
57  public interface NamedNodeMap {
58      /**
59       * Retrieves a node specified by name.
60       * @param name The nodeName of a node to retrieve.
61       * @return A Node (of any type) with the specified
62       *         nodeName, or null if it does not identify

```

```

62     * any node in this map.
63     */
64     public Node getNamedItem(String name);
65
66     /**
67     * Adds a node using its <code>nodeName</code> attribute. If a node with
68     * that name is already present in this map, it is replaced by the new
69     * one. Replacing a node by itself has no effect.
70     * <br>As the <code>nodeName</code> attribute is used to derive the name
71     * which the node must be stored under, multiple nodes of certain types
72     * (those that have a "special" string value) cannot be stored as the
73     * names would clash. This is seen as preferable to allowing nodes to be
74     * aliased.
75     * @param arg A node to store in this map. The node will later be
76     * accessible using the value of its <code>nodeName</code> attribute.
77     * @return If the new <code>Node</code> replaces an existing node the
78     * replaced <code>Node</code> is returned, otherwise <code>null</code>
79     * is returned.
80     * @exception DOMException
81     * WRONG_DOCUMENT_ERR: Raised if <code>arg</code> was created from a
82     * different document than the one that created this map.
83     * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.
84     * <br>INUSE_ATTRIBUTE_ERR: Raised if <code>arg</code> is an
85     * <code>Attr</code> that is already an attribute of another
86     * <code>Element</code> object. The DOM user must explicitly clone
87     * <code>Attr</code> nodes to re-use them in other elements.
88     * <br>HIERARCHY_REQUEST_ERR: Raised if an attempt is made to add a node
89     * doesn't belong in this NamedNodeMap. Examples would include trying
90     * to insert something other than an Attr node into an Element's map
91     * of attributes, or a non-Entity node into the DocumentType's map of
92     * Entities.
93     */
94     public Node setNamedItem(Node arg)
95         throws DOMException;
96
97     /**
98     * Removes a node specified by name. When this map contains the attributes
99     * attached to an element, if the removed attribute is known to have a
100    * default value, an attribute immediately appears containing the
101    * default value as well as the corresponding namespace URI, local name,
102    * and prefix when applicable.
103    * @param name The <code>nodeName</code> of the node to remove.
104    * @return The node removed from this map if a node with such a name
105    * exists.
106    * @exception DOMException
107    * NOT_FOUND_ERR: Raised if there is no node named <code>name</code> in
108    * this map.
109    * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.
110    */
111    public Node removeNamedItem(String name)
112        throws DOMException;
113
114    /**

```

```

115     * Returns the indexth item in the map. If index
116     * is greater than or equal to the number of nodes in this map, this
117     * returns null.
118     * @param index Index into this map.
119     * @return The node at the indexth position in the map, or
120     *         null if that is not a valid index.
121     */
122     public Node item(int index);
123
124     /**
125     * The number of nodes in this map. The range of valid child node indices
126     * is 0 to length-1 inclusive.
127     */
128     public int getLength();
129
130     /**
131     * Retrieves a node specified by local name and namespace URI.
132     *   
Per [Node \(of any type\) with the specified local
138     \*         name and namespace URI, or null if they do not
139     \*         identify any node in this map.
140     \* @exception DOMException
141     \*         NOT\_SUPPORTED\_ERR: May be raised if the implementation does not
142     \*         support the feature "XML" and the language exposed through the
143     \*         Document does not support XML Namespaces \(such as \[namespaceURI and
153     \\* localName. If a node with that namespace URI and that
154     \\* local name is already present in this map, it is replaced by the new
155     \\* one. Replacing a node by itself has no effect.
156     \\*   
Per \\[namespaceURI and
161     \\\*         localName attributes.
162     \\\* @return If the new Node replaces an existing node the
163     \\\*         replaced Node is returned, otherwise null
164     \\\*         is returned.
165     \\\* @exception DOMException
166     \\\*         WRONG\\\_DOCUMENT\\\_ERR: Raised if arg was created from a
167     \\\*         different document than the one that created this map.

```

```

167 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.
168 * <br>INUSE_ATTRIBUTE_ERR: Raised if <code>arg</code> is an
169 * <code>Attr</code> that is already an attribute of another
170 * <code>Element</code> object. The DOM user must explicitly clone
171 * <code>Attr</code> nodes to re-use them in other elements.
172 * <br>HIERARCHY_REQUEST_ERR: Raised if an attempt is made to add a node
173 * doesn't belong in this NamedNodeMap. Examples would include trying
174 * to insert something other than an Attr node into an Element's map
175 * of attributes, or a non-Entity node into the DocumentType's map of
176 * Entities.
177 * <br>NOT_SUPPORTED_ERR: May be raised if the implementation does not
178 * support the feature "XML" and the language exposed through the
179 * Document does not support XML Namespaces (such as [

```

## 36.22 Node.java

Secuencia 353: org/w3c/dom/Node.java

```
1 /*
```

```
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45  * The Node interface is the primary datatype for the entire
46  * Document Object Model. It represents a single node in the document tree.
47  * While all objects implementing the Node interface expose
48  * methods for dealing with children, not all objects implementing the
49  * Node interface may have children. For example,
50  * Text nodes may not have children, and adding children to
51  * such nodes results in a DOMException being raised.
52  * 

The attributes nodeName, nodeValue and
53  * attributes are included as a mechanism to get at node
54  * information without casting down to the specific derived interface. In


```

```

55 * cases where there is no obvious mapping of these attributes for a
56 * specific nodeType (e.g., nodeValue for an
57 * Element or attributes for a Comment
58 * ), this returns null. Note that the specialized interfaces
59 * may contain additional and more convenient mechanisms to get and set the
60 * relevant information.
61 * 

The values of nodeName,
62 * nodeValue, and attributes vary according to the
63 * node type as follows:


64 * 




```



```

108 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
109 * </tr>
110 * <tr>
111 * <td valign='top' rowspan='1' colspan='1'><code>DocumentType</code></td>
112 * <td valign='top' rowspan='1' colspan='1'>same as
113 * <code>DocumentType.name</code></td>
114 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
115 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
116 * </tr>
117 * <tr>
118 * <td valign='top' rowspan='1' colspan='1'>
119 * <code>Element</code></td>
120 * <td valign='top' rowspan='1' colspan='1'>same as <code>Element.tagName</code></td>
121 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
122 * <td valign='top' rowspan='1' colspan='1'>
123 * <code>NamedNodeMap</code></td>
124 * </tr>
125 * <tr>
126 * <td valign='top' rowspan='1' colspan='1'><code>Entity</code></td>
127 * <td valign='top' rowspan='1' colspan='1'>entity name</td>
128 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
129 * <td valign='top' rowspan='1' colspan='1'>
130 * <code>null</code></td>
131 * </tr>
132 * <tr>
133 * <td valign='top' rowspan='1' colspan='1'><code>EntityReference</code></td>
134 * <td valign='top' rowspan='1' colspan='1'>name of entity referenced</td>
135 * <td valign='top' rowspan='1' colspan='1'>
136 * <code>null</code></td>
137 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
138 * </tr>
139 * <tr>
140 * <td valign='top' rowspan='1' colspan='1'><code>Notation</code></td>
141 * <td valign='top' rowspan='1' colspan='1'>notation name</td>
142 * <td valign='top' rowspan='1' colspan='1'>
143 * <code>null</code></td>
144 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
145 * </tr>
146 * <tr>
147 * <td valign='top' rowspan='1' colspan='1'><code>ProcessingInstruction</code></td>
148 * <td valign='top' rowspan='1' colspan='1'>same
149 * as <code>ProcessingInstruction.target</code></td>
150 * <td valign='top' rowspan='1' colspan='1'>same as
151 * <code>ProcessingInstruction.data</code></td>
152 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>
153 * </tr>
154 * <tr>
155 * <td valign='top' rowspan='1' colspan='1'><code>Text</code></td>
156 * <td valign='top' rowspan='1' colspan='1'>
157 * <code>"#text"</code></td>
158 * <td valign='top' rowspan='1' colspan='1'>same as <code>CharacterData.data</code>, the content
159 * of the text node</td>
160 * <td valign='top' rowspan='1' colspan='1'><code>null</code></td>

```

```

161 * </tr>
162 * </table>
163 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
164 */
165 public interface Node {
166     // NodeType
167     /**
168      * The node is an <code>Element</code>.
169      */
170     public static final short ELEMENT_NODE          = 1;
171     /**
172      * The node is an <code>Attr</code>.
173      */
174     public static final short ATTRIBUTE_NODE        = 2;
175     /**
176      * The node is a <code>Text</code> node.
177      */
178     public static final short TEXT_NODE             = 3;
179     /**
180      * The node is a <code>CDATASection</code>.
181      */
182     public static final short CDATA_SECTION_NODE    = 4;
183     /**
184      * The node is an <code>EntityReference</code>.
185      */
186     public static final short ENTITY_REFERENCE_NODE = 5;
187     /**
188      * The node is an <code>Entity</code>.
189      */
190     public static final short ENTITY_NODE           = 6;
191     /**
192      * The node is a <code>ProcessingInstruction</code>.
193      */
194     public static final short PROCESSING_INSTRUCTION_NODE = 7;
195     /**
196      * The node is a <code>Comment</code>.
197      */
198     public static final short COMMENT_NODE          = 8;
199     /**
200      * The node is a <code>Document</code>.
201      */
202     public static final short DOCUMENT_NODE         = 9;
203     /**
204      * The node is a <code>DocumentType</code>.
205      */
206     public static final short DOCUMENT_TYPE_NODE    = 10;
207     /**
208      * The node is a <code>DocumentFragment</code>.
209      */
210     public static final short DOCUMENT_FRAGMENT_NODE = 11;
211     /**
212      * The node is a <code>Notation</code>.

```

```

213     */
214     public static final short NOTATION_NODE           = 12;
215
216     /**
217      * The name of this node, depending on its type; see the table above.
218      */
219     public String getNodeName();
220
221     /**
222      * The value of this node, depending on its type; see the table above.
223      * When it is defined to be null, setting it has no effect,
224      * including if the node is read-only.
225      * @exception DOMException
226      *     DOMSTRING_SIZE_ERR: Raised when it would return more characters than
227      *     fit in a DOMString variable on the implementation
228      *     platform.
229      */
230     public String getNodeValue()
231         throws DOMException;
232
233     /**
234      * The value of this node, depending on its type; see the table above.
235      * When it is defined to be null, setting it has no effect,
236      * including if the node is read-only.
237      * @exception DOMException
238      *     NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly and if
239      *     it is not defined to be null.
240      */
241     public void setNodeValue(String nodeValue)
242         throws DOMException;
243
244     /**
245      * A code representing the type of the underlying object, as defined above.
246      */
247     public short getNodeType();
248
249     /**
250      * The parent of this node. All nodes, except Attr,
251      * Document, DocumentFragment,
252      * Entity, and Notation may have a parent.
253      * However, if a node has just been created and not yet added to the
254      * tree, or if it has been removed from the tree, this is
255      * null.
256      */
257     public Node getParentNode();
258
259     /**
260      * A NodeList that contains all children of this node. If
261      * there are no children, this is a NodeList containing no
262      * nodes.
263      */
264     public NodeList getChildNodes();
265
266     /**

```

```
266     * The first child of this node. If there is no such node, this returns
267     * <code>null</code>.
268     */
269     public Node getFirstChild();
270
271     /**
272     * The last child of this node. If there is no such node, this returns
273     * <code>null</code>.
274     */
275     public Node getLastChild();
276
277     /**
278     * The node immediately preceding this node. If there is no such node,
279     * this returns <code>null</code>.
280     */
281     public Node getPreviousSibling();
282
283     /**
284     * The node immediately following this node. If there is no such node,
285     * this returns <code>null</code>.
286     */
287     public Node getNextSibling();
288
289     /**
290     * A <code>NamedNodeMap</code> containing the attributes of this node (if
291     * it is an <code>Element</code>) or <code>null</code> otherwise.
292     */
293     public NamedNodeMap getAttributes();
294
295     /**
296     * The <code>Document</code> object associated with this node. This is
297     * also the <code>Document</code> object used to create new nodes. When
298     * this node is a <code>Document</code> or a <code>DocumentType</code>
299     * which is not used with any <code>Document</code> yet, this is
300     * <code>null</code>.
301     *
302     * @since DOM Level 2
303     */
304     public Document getOwnerDocument();
305
306     /**
307     * Inserts the node <code>newChild</code> before the existing child node
308     * <code>refChild</code>. If <code>refChild</code> is <code>null</code>,
309     * insert <code>newChild</code> at the end of the list of children.
310     * <br>If <code>newChild</code> is a <code>DocumentFragment</code> object,
311     * all of its children are inserted, in the same order, before
312     * <code>refChild</code>. If the <code>newChild</code> is already in the
313     * tree, it is first removed.
314     * <p><b>Note:</b> Inserting a node before itself is implementation
315     * dependent.
316     * @param newChild The node to insert.
317     * @param refChild The reference node, i.e., the node before which the
318     *     new node must be inserted.
```

```

319     * @return The node being inserted.
320     * @exception DOMException
321     *     HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not
322     *     allow children of the type of the <code>newChild</code> node, or if
323     *     the node to insert is one of this node's ancestors or this node
324     *     itself, or if this node is of type <code>Document</code> and the
325     *     DOM application attempts to insert a second
326     *     <code>DocumentType</code> or <code>Element</code> node.
327     *     <br>WRONG_DOCUMENT_ERR: Raised if <code>newChild</code> was created
328     *     from a different document than the one that created this node.
329     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly or
330     *     if the parent of the node being inserted is readonly.
331     *     <br>NOT_FOUND_ERR: Raised if <code>refChild</code> is not a child of
332     *     this node.
333     *     <br>NOT_SUPPORTED_ERR: if this node is of type <code>Document</code>,
334     *     this exception might be raised if the DOM implementation doesn't
335     *     support the insertion of a <code>DocumentType</code> or
336     *     <code>Element</code> node.
337     *
338     * @since DOM Level 3
339     */
340     public Node insertBefore(Node newChild,
341                             Node refChild)
342         throws DOMException;
343
344     /**
345     * Replaces the child node <code>oldChild</code> with <code>newChild</code>
346     * in the list of children, and returns the <code>oldChild</code> node.
347     * <br>If <code>newChild</code> is a <code>DocumentFragment</code> object,
348     * <code>oldChild</code> is replaced by all of the
349     * <code>DocumentFragment</code> children, which are inserted in the
350     * same order. If the <code>newChild</code> is already in the tree, it
351     * is first removed.
352     * <p><b>Note:</b> Replacing a node with itself is implementation
353     * dependent.
354     * @param newChild The new node to put in the child list.
355     * @param oldChild The node being replaced in the list.
356     * @return The node replaced.
357     * @exception DOMException
358     *     HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not
359     *     allow children of the type of the <code>newChild</code> node, or if
360     *     the node to put in is one of this node's ancestors or this node
361     *     itself, or if this node is of type <code>Document</code> and the
362     *     result of the replacement operation would add a second
363     *     <code>DocumentType</code> or <code>Element</code> on the
364     *     <code>Document</code> node.
365     *     <br>WRONG_DOCUMENT_ERR: Raised if <code>newChild</code> was created
366     *     from a different document than the one that created this node.
367     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this node or the parent of
368     *     the new node is readonly.
369     *     <br>NOT_FOUND_ERR: Raised if <code>oldChild</code> is not a child of
370     *     this node.
371     *     <br>NOT_SUPPORTED_ERR: if this node is of type <code>Document</code>,

```

```

372 * this exception might be raised if the DOM implementation doesn't
373 * support the replacement of the <code>DocumentType</code> child or
374 * <code>Element</code> child.
375 *
376 * @since DOM Level 3
377 */
378 public Node replaceChild(Node newChild,
379                          Node oldChild)
380     throws DOMException;
381
382 /**
383  * Removes the child node indicated by <code>oldChild</code> from the list
384  * of children, and returns it.
385  * @param oldChild The node being removed.
386  * @return The node removed.
387  * @exception DOMException
388  *   NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
389  *   <br>NOT_FOUND_ERR: Raised if <code>oldChild</code> is not a child of
390  *   this node.
391  *   <br>NOT_SUPPORTED_ERR: if this node is of type <code>Document</code>,
392  *   this exception might be raised if the DOM implementation doesn't
393  *   support the removal of the <code>DocumentType</code> child or the
394  *   <code>Element</code> child.
395  *
396  * @since DOM Level 3
397  */
398 public Node removeChild(Node oldChild)
399     throws DOMException;
400
401 /**
402  * Adds the node <code>newChild</code> to the end of the list of children
403  * of this node. If the <code>newChild</code> is already in the tree, it
404  * is first removed.
405  * @param newChild The node to add. If it is a
406  *   <code>DocumentFragment</code> object, the entire contents of the
407  *   document fragment are moved into the child list of this node
408  * @return The node added.
409  * @exception DOMException
410  *   HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not
411  *   allow children of the type of the <code>newChild</code> node, or if
412  *   the node to append is one of this node's ancestors or this node
413  *   itself, or if this node is of type <code>Document</code> and the
414  *   DOM application attempts to append a second
415  *   <code>DocumentType</code> or <code>Element</code> node.
416  *   <br>WRONG_DOCUMENT_ERR: Raised if <code>newChild</code> was created
417  *   from a different document than the one that created this node.
418  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly or
419  *   if the previous parent of the node being inserted is readonly.
420  *   <br>NOT_SUPPORTED_ERR: if the <code>newChild</code> node is a child
421  *   of the <code>Document</code> node, this exception might be raised
422  *   if the DOM implementation doesn't support the removal of the
423  *   <code>DocumentType</code> child or <code>Element</code> child.
424  *

```

```

425     * @since DOM Level 3
426     */
427     public Node appendChild(Node newChild)
428         throws DOMException;
429
430     /**
431     * Returns whether this node has any children.
432     * @return Returns true if this node has any children,
433     *         false otherwise.
434     */
435     public boolean hasChildNodes();
436
437     /**
438     * Returns a duplicate of this node, i.e., serves as a generic copy
439     * constructor for nodes. The duplicate node has no parent (
440     * parentNode is null) and no user data. User
441     * data associated to the imported node is not carried over. However, if
442     * any UserDataHandlers has been specified along with the
443     * associated data these handlers will be called with the appropriate
444     * parameters before this method returns.
445     *   
Cloning an Element copies all attributes and their
446     * values, including those generated by the XML processor to represent
447     * defaulted attributes, but this method does not copy any children it
448     * contains unless it is a deep clone. This includes text contained in
449     * an the Element since the text is contained in a child
450     * Text node. Cloning an Attr directly, as
451     * opposed to be cloned as part of an Element cloning
452     * operation, returns a specified attribute (specified is
453     * true). Cloning an Attr always clones its
454     * children, since they represent its value, no matter whether this is a
455     * deep clone or not. Cloning an EntityReference
456     * automatically constructs its subtree if a corresponding
457     * Entity is available, no matter whether this is a deep
458     * clone or not. Cloning any other type of node simply returns a copy of
459     * this node.
460     *   
Note that cloning an immutable subtree results in a mutable copy,
461     * but the children of an EntityReference clone are readonly
462     * . In addition, clones of unspecified Attr nodes are
463     * specified. And, cloning Document,
464     * DocumentType, Entity, and
465     * Notation nodes is implementation dependent.
466     * @param deep If true, recursively clone the subtree under
467     *         the specified node; if false, clone only the node
468     *         itself (and its attributes, if it is an Element).
469     * @return The duplicate node.
470     */
471     public Node cloneNode(boolean deep);
472
473     /**
474     * Puts all Text nodes in the full depth of the sub-tree
475     * underneath this Node, including attribute nodes, into a
476     * "normal" form where only structure (e.g., elements, comments,
477     * processing instructions, CDATA sections, and entity references)

```



```

478 * separates <code>Text</code> nodes, i.e., there are neither adjacent
479 * <code>Text</code> nodes nor empty <code>Text</code> nodes. This can
480 * be used to ensure that the DOM view of a document is the same as if
481 * it were saved and re-loaded, and is useful when operations (such as
482 * XPointer [

```



```

530 /**
531  * The namespace prefix of this node, or null if it is
532  * unspecified. When it is defined to be null, setting it
533  * has no effect, including if the node is read-only.
534  *   
Note that setting this attribute, when permitted, changes the
535  * nodeName attribute, which holds the qualified name, as
536  * well as the tagName and name attributes of
537  * the Element and Attr interfaces, when
538  * applicable.
539  *   
Setting the prefix to null makes it unspecified,
540  * setting it to an empty string is implementation dependent.
541  *   
Note also that changing the prefix of an attribute that is known to
542  * have a default value, does not make a new attribute with the default
543  * value and the original prefix appear, since the
544  * namespaceURI and localName do not change.
545  *   
For nodes of any type other than ELEMENT_NODE and
546  * ATTRIBUTE_NODE and nodes created with a DOM Level 1
547  * method, such as createElement from the
548  * Document interface, this is always null.
549  *
550  * @since DOM Level 2
551  */
552 public String getPrefix();
553 /**
554  * The namespace prefix of this node, or null if it is
555  * unspecified. When it is defined to be null, setting it
556  * has no effect, including if the node is read-only.
557  *   
Note that setting this attribute, when permitted, changes the
558  * nodeName attribute, which holds the qualified name, as
559  * well as the tagName and name attributes of
560  * the Element and Attr interfaces, when
561  * applicable.
562  *   
Setting the prefix to null makes it unspecified,
563  * setting it to an empty string is implementation dependent.
564  *   
Note also that changing the prefix of an attribute that is known to
565  * have a default value, does not make a new attribute with the default
566  * value and the original prefix appear, since the
567  * namespaceURI and localName do not change.
568  *   
For nodes of any type other than ELEMENT_NODE and
569  * ATTRIBUTE_NODE and nodes created with a DOM Level 1
570  * method, such as createElement from the
571  * Document interface, this is always null.
572  * @exception DOMException
573  *   INVALID_CHARACTER_ERR: Raised if the specified prefix contains an
574  *   illegal character according to the XML version in use specified in
575  *   the Document.xmlVersion attribute.
576  *     
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
577  *     
NAMESPACE_ERR: Raised if the specified prefix is
578  *   malformed per the Namespaces in XML specification, if the
579  *   namespaceURI of this node is null, if the
580  *   specified prefix is "xml" and the namespaceURI of this
581  *   node is different from "<a href='http://www.w3.org/XML/1998/namespace'>
582  *   http://www.w3.org/XML/1998/namespace</a>", if this node is an attribute and the specified

```

```

    prefix is "xmlns" and
583 * the <code>namespaceURI</code> of this node is different from "<a href='http://www.w3.org
    /2000/xmlns/'>http://www.w3.org/2000/xmlns/</a>", or if this node is an attribute and the
    <code>qualifiedName</code> of
584 * this node is "xmlns" [<a href='http://www.w3.org/TR/1999/REC-xml-names-19990114/'>XML
    Namespaces</a>]
585 * .
586 *
587 * @since DOM Level 2
588 */
589 public void setPrefix(String prefix)
590     throws DOMException;
591
592 /**
593 * Returns the local part of the qualified name of this node.
594 * <br>For nodes of any type other than <code>ELEMENT_NODE</code> and
595 * <code>ATTRIBUTE_NODE</code> and nodes created with a DOM Level 1
596 * method, such as <code>Document.createElement()</code>, this is always
597 * <code>null</code>.
598 *
599 * @since DOM Level 2
600 */
601 public String getLocalName();
602
603 /**
604 * Returns whether this node (if it is an element) has any attributes.
605 * @return Returns <code>true</code> if this node has any attributes,
606 * <code>false</code> otherwise.
607 *
608 * @since DOM Level 2
609 */
610 public boolean hasAttributes();
611
612 /**
613 * The absolute base URI of this node or <code>null</code> if the
614 * implementation wasn't able to obtain an absolute URI. This value is
615 * computed as described in . However, when the <code>Document</code>
616 * supports the feature "HTML" [<a href='http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML
    -20030109'>DOM Level 2 HTML</a>]
617 * , the base URI is computed using first the value of the href
618 * attribute of the HTML BASE element if any, and the value of the
619 * <code>documentURI</code> attribute from the <code>Document</code>
620 * interface otherwise.
621 *
622 * @since DOM Level 3
623 */
624 public String getBaseURI();
625
626 // DocumentPosition
627 /**
628 * The two nodes are disconnected. Order between disconnected nodes is
629 * always implementation-specific.
630 */

```

```

631 public static final short DOCUMENT_POSITION_DISCONNECTED = 0x01;
632 /**
633  * The second node precedes the reference node.
634  */
635 public static final short DOCUMENT_POSITION_PRECEDING = 0x02;
636 /**
637  * The node follows the reference node.
638  */
639 public static final short DOCUMENT_POSITION_FOLLOWING = 0x04;
640 /**
641  * The node contains the reference node. A node which contains is always
642  * preceding, too.
643  */
644 public static final short DOCUMENT_POSITION_CONTAINS = 0x08;
645 /**
646  * The node is contained by the reference node. A node which is contained
647  * is always following, too.
648  */
649 public static final short DOCUMENT_POSITION_CONTAINED_BY = 0x10;
650 /**
651  * The determination of preceding versus following is
652  * implementation-specific.
653  */
654 public static final short DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC = 0x20;
655
656 /**
657  * Compares the reference node, i.e. the node on which this method is
658  * being called, with a node, i.e. the one passed as a parameter, with
659  * regard to their position in the document and according to the
660  * document order.
661  * @param other The node to compare against the reference node.
662  * @return Returns how the node is positioned relatively to the reference
663  *         node.
664  * @exception DOMException
665  *         NOT_SUPPORTED_ERR: when the compared nodes are from different DOM
666  *         implementations that do not coordinate to return consistent
667  *         implementation-specific results.
668  *
669  * @since DOM Level 3
670  */
671 public short compareDocumentPosition(Node other)
672                                     throws DOMException;
673
674 /**
675  * This attribute returns the text content of this node and its
676  * descendants. When it is defined to be null, setting it
677  * has no effect. On setting, any possible children this node may have
678  * are removed and, if it the new string is not empty or
679  * null, replaced by a single Text node
680  * containing the string this attribute is set to.
681  * <br> On getting, no serialization is performed, the returned string
682  * does not contain any markup. No whitespace normalization is performed
683  * and the returned string does not contain the white spaces in element

```

```

684 * content (see the attribute
685 * Text.isElementContentWhitespace). Similarly, on setting,
686 * no parsing is performed either, the input string is taken as pure
687 * textual content.
688 *   
The string returned is made of the text content of this node
689 * depending on its type, as defined below:
690 * 




```

```

737     * textual content.
738     * <br>The string returned is made of the text content of this node
739     * depending on its type, as defined below:
740     * <table border='1' cellpadding='3'>
741     * <tr>
742     * <th>Node type</th>
743     * <th>Content</th>
744     * </tr>
745     * <tr>
746     * <td valign='top' rowspan='1' colspan='1'>
747     * ELEMENT_NODE, ATTRIBUTE_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE,
748     * DOCUMENT_FRAGMENT_NODE</td>
749     * <td valign='top' rowspan='1' colspan='1'>concatenation of the <code>textContent</code>
750     * attribute value of every child node, excluding COMMENT_NODE and
751     * PROCESSING_INSTRUCTION_NODE nodes. This is the empty string if the
752     * node has no children.</td>
753     * </tr>
754     * <tr>
755     * <td valign='top' rowspan='1' colspan='1'>TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE,
756     * PROCESSING_INSTRUCTION_NODE</td>
757     * <td valign='top' rowspan='1' colspan='1'><code>nodeValue</code></td>
758     * </tr>
759     * <tr>
760     * <td valign='top' rowspan='1' colspan='1'>DOCUMENT_NODE,
761     * DOCUMENT_TYPE_NODE, NOTATION_NODE</td>
762     * <td valign='top' rowspan='1' colspan='1'><em>null</em></td>
763     * </tr>
764     * </table>
765     * @exception DOMException
766     *     NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.
767     *
768     * @since DOM Level 3
769     */
770     public void settextContent(String textContent)
771                               throws DOMException;
772
773     /**
774     * Returns whether this node is the same node as the given one.
775     * <br>This method provides a way to determine whether two
776     * <code>Node</code> references returned by the implementation reference
777     * the same object. When two <code>Node</code> references are references
778     * to the same object, even if through a proxy, the references may be
779     * used completely interchangeably, such that all attributes have the
780     * same values and calling the same DOM method on either reference
781     * always has exactly the same effect.
782     * @param other The node to test against.
783     * @return Returns <code>true</code> if the nodes are the same,
784     *         <code>false</code> otherwise.
785     *
786     * @since DOM Level 3
787     */
788     public boolean isSameNode(Node other);
789

```

```

790 /**
791  * Look up the prefix associated to the given namespace URI, starting from
792  * this node. The default namespace declarations are ignored by this
793  * method.
794  * <br>See for details on the algorithm used by this method.
795  * @param namespaceURI The namespace URI to look for.
796  * @return Returns an associated namespace prefix if found or
797  * <code>null</code> if none is found. If more than one prefix are
798  * associated to the namespace prefix, the returned namespace prefix
799  * is implementation dependent.
800  *
801  * @since DOM Level 3
802  */
803 public String lookupPrefix(String namespaceURI);
804
805 /**
806  * This method checks if the specified <code>namespaceURI</code> is the
807  * default namespace or not.
808  * @param namespaceURI The namespace URI to look for.
809  * @return Returns <code>true</code> if the specified
810  * <code>namespaceURI</code> is the default namespace,
811  * <code>false</code> otherwise.
812  *
813  * @since DOM Level 3
814  */
815 public boolean isDefaultNamespace(String namespaceURI);
816
817 /**
818  * Look up the namespace URI associated to the given prefix, starting from
819  * this node.
820  * <br>See for details on the algorithm used by this method.
821  * @param prefix The prefix to look for. If this parameter is
822  * <code>null</code>, the method will return the default namespace URI
823  * if any.
824  * @return Returns the associated namespace URI or <code>null</code> if
825  * none is found.
826  *
827  * @since DOM Level 3
828  */
829 public String lookupNamespaceURI(String prefix);
830
831 /**
832  * Tests whether two nodes are equal.
833  * <br>This method tests for equality of nodes, not sameness (i.e.,
834  * whether the two nodes are references to the same object) which can be
835  * tested with <code>Node.isSameNode()</code>. All nodes that are the
836  * same will also be equal, though the reverse may not be true.
837  * <br>Two nodes are equal if and only if the following conditions are
838  * satisfied:
839  * <ul>
840  * <li>The two nodes are of the same type.
841  * </li>
842  * <li>The following string

```

```

843 * attributes are equal: nodeName, localName,
844 * namespaceURI, prefix, nodeValue
845 * . This is: they are both null, or they have the same
846 * length and are character for character identical.
847 * </li>
848 * <li>The
849 * <code>attributes</code> <code>NamedNodeMaps</code> are equal. This
850 * is: they are both null, or they have the same length and
851 * for each node that exists in one map there is a node that exists in
852 * the other map and is equal, although not necessarily at the same
853 * index.
854 * </li>
855 * <li>The <code>childNodes</code> <code>NodeLists</code> are equal.
856 * This is: they are both null, or they have the same
857 * length and contain equal nodes at the same index. Note that
858 * normalization can affect equality; to avoid this, nodes should be
859 * normalized before being compared.
860 * </li>
861 * </ul>
862 * <br>For two <code>DocumentType</code> nodes to be equal, the following
863 * conditions must also be satisfied:
864 * <ul>
865 * <li>The following string attributes
866 * are equal: <code>publicId</code>, <code>systemId</code>,
867 * <code>internalSubset</code>.
868 * </li>
869 * <li>The <code>entities</code>
870 * <code>NamedNodeMaps</code> are equal.
871 * </li>
872 * <li>The <code>notations</code>
873 * <code>NamedNodeMaps</code> are equal.
874 * </li>
875 * </ul>
876 * <br>On the other hand, the following do not affect equality: the
877 * <code>ownerDocument</code>, <code>baseURI</code>, and
878 * <code>parentNode</code> attributes, the <code>specified</code>
879 * attribute for <code>Attr</code> nodes, the <code>schemaTypeInfo</code>
880 * attribute for <code>Attr</code> and <code>Element</code> nodes, the
881 * <code>Text.isElementContentWhitespace</code> attribute for
882 * <code>Text</code> nodes, as well as any user data or event listeners
883 * registered on the nodes.
884 * <p><b>Note:</b> As a general rule, anything not mentioned in the
885 * description above is not significant in consideration of equality
886 * checking. Note that future versions of this specification may take
887 * into account more attributes and implementations conform to this
888 * specification are expected to be updated accordingly.
889 * @param arg The node to compare equality with.
890 * @return Returns <code>true</code> if the nodes are equal,
891 * <code>false</code> otherwise.
892 *
893 * @since DOM Level 3
894 */
895 public boolean isEqualNode(Node arg);

```



```

896
897 /**
898  * This method returns a specialized object which implements the
899  * specialized APIs of the specified feature and version, as specified
900  * in . The specialized object may also be obtained by using
901  * binding-specific casting methods but is not necessarily expected to,
902  * as discussed in . This method also allow the implementation to
903  * provide specialized objects which do not support the Node
904  * interface.
905  * @param feature The name of the feature requested. Note that any plus
906  * sign "+" prepended to the name of the feature will be ignored since
907  * it is not significant in the context of this method.
908  * @param version This is the version number of the feature to test.
909  * @return Returns an object which implements the specialized APIs of
910  * the specified feature and version, if any, or null if
911  * there is no object which implements interfaces associated with that
912  * feature. If the DOMObject returned by this method
913  * implements the Node interface, it must delegate to the
914  * primary core Node and not return results inconsistent
915  * with the primary core Node such as attributes,
916  * childNodes, etc.
917  *
918  * @since DOM Level 3
919  */
920 public Object getFeature(String feature,
921                        String version);
922
923 /**
924  * Associate an object to a key on this node. The object can later be
925  * retrieved from this node by calling getUserData with the
926  * same key.
927  * @param key The key to associate the object to.
928  * @param data The object to associate to the given key, or
929  * null to remove any existing association to that key.
930  * @param handler The handler to associate to that key, or
931  * null.
932  * @return Returns the DOMUserData previously associated to
933  * the given key on this node, or null if there was none.
934  *
935  * @since DOM Level 3
936  */
937 public Object setUserData(String key,
938                        Object data,
939                        UserDataHandler handler);
940
941 /**
942  * Retrieves the object associated to a key on a this node. The object
943  * must first have been set to this node by calling
944  * setUserData with the same key.
945  * @param key The key the object is associated to.
946  * @return Returns the DOMUserData associated to the given
947  * key on this node, or null if there was none.
948  *

```



```
949     * @since DOM Level 3
950     */
951     public Object getUserData(String key);
952
953 }
```

### 36.23 NodeList.java

Secuencia 354: org/w3c/dom/NodeList.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
```

```

45  * The NodeList interface provides the abstraction of an ordered
46  * collection of nodes, without defining or constraining how this collection
47  * is implemented. NodeList objects in the DOM are live.
48  * 

The items in the NodeList are accessible via an integral
49  * index, starting from 0.


50  * 

See also the Document
    Object Model (DOM) Level 3 Core Specification.


51  */
52  public interface NodeList {
53      /**
54       * Returns the indexth item in the collection. If
55       * index is greater than or equal to the number of nodes in
56       * the list, this returns null.
57       * @param index Index into the collection.
58       * @return The node at the indexth position in the
59       *         NodeList, or null if that is not a valid
60       *         index.
61       */
62      public Node item(int index);
63
64      /**
65       * The number of nodes in the list. The range of valid child node indices
66       * is 0 to length-1 inclusive.
67       */
68      public int getLength();
69
70  }

```

## 36.24 Notation.java

Secuencia 355: org/w3c/dom/Notation.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *

```

```

23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * This interface represents a notation declared in the DTD. A notation either
46   * declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification [XML 1.0]), or is
47   * used for formal declaration of processing instruction targets (see section 2.6 of the XML 1.0 specification [XML 1.0]). The
48   * nodeName attribute inherited from Node is set
49   * to the declared name of the notation.
50   * 

The DOM Core does not support editing Notation nodes; they
51   * are therefore readonly.


52   * 

A Notation node does not have any parent.


53   * 

See also the Document Object Model \(DOM\) Level 3 Core Specification.


54   */
55  public interface Notation extends Node {
56      /**
57       * The public identifier of this notation. If the public identifier was
58       * not specified, this is null.
59       */
60      public String getPublicId();
61
62      /**
63       * The system identifier of this notation. If the system identifier was
64       * not specified, this is null. This may be an absolute URI
65       * or not.
66       */
67      public String getSystemId();
68
69  }

```

**36.25** ProcessingInstruction.java

Secuencia 356: org/w3c/dom/ProcessingInstruction.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45 * The ProcessingInstruction interface represents a "processing
46 * instruction", used in XML as a way to keep processor-specific information
47 * in the text of the document.
48 * <p> No lexical check is done on the content of a processing instruction and
49 * it is therefore possible to have the character sequence
50 * "?>" in the content, which is illegal a processing
```

```

51  * instruction per section 2.6 of [

```

## 36.26 Text.java

Secuencia 357: org/w3c/dom/Text.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```

18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom;
43
44  /**
45   * The Text interface inherits from CharacterData
46   * and represents the textual content (termed character data in XML) of an Element or Attr. If there
      is no
47   * markup inside an element's content, the text is contained in a single
48   * object implementing the Text interface that is the only
49   * child of the element. If there is markup, it is parsed into the
50   * information items (elements, comments, etc.) and Text nodes
51   * that form the list of children of the element.
52   * 

When a document is first made available via the DOM, there is only one
53   * Text node for each block of text. Users may create adjacent
54   * Text nodes that represent the contents of a given element
55   * without any intervening markup, but should be aware that there is no way
56   * to represent the separations between these nodes in XML or HTML, so they
57   * will not (in general) persist between DOM editing sessions. The
58   * Node.normalize() method merges any such adjacent
59   * Text objects into a single node for each block of text.
60   * 

No lexical check is done on the content of a Text node
61   * and, depending on its position in the document, some characters must be
62   * escaped during serialization using character references; e.g. the
63   * characters "&" if the textual content is part of an element or of
64   * an attribute, the character sequence "]]>" when part of an element,
65   * the quotation mark character " or the apostrophe character ' when part of
66   * an attribute.
67   * 

See also the Document
      Object Model \(DOM\) Level 3 Core Specification.


```

```

68  */
69  public interface Text extends CharacterData {
70      /**
71       * Breaks this node into two nodes at the specified <code>offset</code>,
72       * keeping both in the tree as siblings. After being split, this node
73       * will contain all the content up to the <code>offset</code> point. A
74       * new node of the same type, which contains all the content at and
75       * after the <code>offset</code> point, is returned. If the original
76       * node had a parent node, the new node is inserted as the next sibling
77       * of the original node. When the <code>offset</code> is equal to the
78       * length of this node, the new node has no data.
79       * @param offset The 16-bit unit offset at which to split, starting from
80       * <code>0</code>.
81       * @return The new node, of the same type as this node.
82       * @exception DOMException
83       *     INDEX_SIZE_ERR: Raised if the specified offset is negative or greater
84       *     than the number of 16-bit units in <code>data</code>.
85       *     NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
86       */
87     public Text splitText(int offset)
88         throws DOMException;
89
90     /**
91      * Returns whether this text node contains <a href='http://www.w3.org/TR/2004/REC-xml-info
92      * set-20040204#infoitem.character'>
93      * element content whitespace</a>, often abusively called "ignorable whitespace". The text node
94      * is
95      * determined to contain whitespace in element content during the load
96      * of the document or if validation occurs while using
97      * <code>Document.normalizeDocument()</code>.
98      * @since DOM Level 3
99      */
100    public boolean isElementContentWhitespace();
101
102    /**
103     * Returns all text of <code>Text</code> nodes logically-adjacent text
104     * nodes to this node, concatenated in document order.
105     * <br>For instance, in the example below <code>wholeText</code> on the
106     * <code>Text</code> node that contains "bar" returns "barfoo", while on
107     * the <code>Text</code> node that contains "foo" it returns "barfoo".
108     *
109     * <pre>
110     *           +-----+
111     *           | <code><code>p</code></code> |
112     *           +-----+
113     *           /\
114     *          /\
115     * /-----\  +-----+
116 * | bar |    | <code><code>&ent</code></code> |
117 * \-----/  +-----+
118 *              |
119 *              |
120 *             /-----\

```

```

119      *          | foo |
120      *          \-----/
121      * </pre>
122      * <em>Figure: barTextNode.wholeText value is "barfoo"</em>
123      *
124      * @since DOM Level 3
125      */
126      public String getWholeText();
127
128      /**
129       * Replaces the text of the current node and all logically-adjacent text
130       * nodes with the specified text. All logically-adjacent text nodes are
131       * removed including the current node unless it was the recipient of the
132       * replacement text.
133       * <p>This method returns the node which received the replacement text.
134       * The returned node is:</p>
135       * <ul>
136       * <li><code>null</code>, when the replacement text is
137       * the empty string;
138       * </li>
139       * <li>the current node, except when the current node is
140       * read-only;
141       * </li>
142       * <li>a new <code>Text</code> node of the same type (
143       * <code>Text</code> or <code>CDATASection</code>) as the current node
144       * inserted at the location of the replacement.
145       * </li>
146       * </ul>
147       * <p>For instance, in the above example calling
148       * <code>replaceWholeText</code> on the <code>Text</code> node that
149       * contains "bar" with "yo" in argument results in the following:</p>
150       *
151       * <pre>
152       *          +-----+
153       *          | &lt;p&gt; |
154       *          +-----+
155       *          |
156       *          |
157       *          /-----\
158       *          | yo |
159       *          \-----/
160       * </pre>
161       * <em>Figure: barTextNode.replaceWholeText("yo") modifies the
162       * textual content of barTextNode with "yo"</em>
163       *
164       * <p>Where the nodes to be removed are read-only descendants of an
165       * <code>EntityReference</code>, the <code>EntityReference</code> must
166       * be removed instead of the read-only nodes. If any
167       * <code>EntityReference</code> to be removed has descendants that are
168       * not <code>EntityReference</code>, <code>Text</code>, or
169       * <code>CDATASection</code> nodes, the <code>replaceWholeText</code>
170       * method must fail before performing any modification of the document,
171       * raising a <code>DOMException</code> with the code

```



```

172 * <code>NO_MODIFICATION_ALLOWED_ERR</code>.</p>
173 * <p>For instance, in the example below calling
174 * <code>replaceWholeText</code> on the <code>Text</code> node that
175 * contains "bar" fails, because the <code>EntityReference</code> node
176 * "ent" contains an <code>Element</code> node which cannot be removed.</p>
177 * @param content The content of the replacing <code>Text</code> node.
178 * @return The <code>Text</code> node created with the specified content.
179 * @exception DOMException
180 *     NO_MODIFICATION_ALLOWED_ERR: Raised if one of the <code>Text</code>
181 *     nodes being replaced is readonly.
182 * @since DOM Level 3
183 */
184 public Text replaceWholeText(String content)
185     throws DOMException;
186
187 }

```

### 36.27 TypeInfo.java

Secuencia 358: org/w3c/dom/TypeInfo.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for

```

```

34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom;
43
44 /**
45 * The TypeInfo interface represents a type referenced from
46 * Element or Attr nodes, specified in the schemas
47 * associated with the document. The type is a pair of a namespace URI and
48 * name properties, and depends on the document's schema.
49 * 

If the document's schema is an XML DTD [http://www.w3.org/TR/2004/REC-xml-20040204], the values
50 * are computed as follows:
51 * 


52 * - If this type is referenced from an
53 * Attr node, typeNamespace is
54 * "http://www.w3.org/TR/REC-xml" and typeName
55 * represents the [attribute type] property in the [http://www.w3.org/TR/2004/REC-xml-info-20040204], the values
56 * . If there is no declaration for the attribute, typeNamespace
57 * and typeName are null.
58 *

59 * - If this type is
60 * referenced from an Element node, typeNamespace
61 * and typeName are null.
62 *

63 * 

64 * 

If the document's schema is an XML Schema [http://www.w3.org/TR/2001/REC-xmlschema-1-20010502], the values
65 * are computed as follows using the post-schema-validation
66 * infoset contributions (also called PSVI contributions):
67 * 


68 * - If the [validity] property exists AND is "invalid" or "notKnown":
69 * the {target namespace} and {name} properties of the declared type if
70 * available, otherwise null.
71 * 

Note: At the time of writing, the XML Schema specification does
72 * not require exposing the declared type. Thus, DOM implementations might
73 * choose not to provide type information if validity is not valid.


74 *

75 * - If the [validity] property exists and is "valid":
76 *

77 *   - If [member type definition] exists:
78 *

79 *     - If {name} is not absent, then expose {name} and {target
80 * namespace} properties of the [member type definition] property;
81 *

82 *     - Otherwise, expose the namespace and local name of the
83 * corresponding anonymous type name.


```

```

83  * </li>
84  * </ol>
85  * </li>
86  * <li> If the <b>[type definition]</b> property exists:
87  * <ol>
88  * <li>If {name} is not absent, then expose {name} and {target
89  * namespace} properties of the <b>[type definition]</b> property;
90  * </li>
91  * <li>Otherwise, expose the namespace and local name of the
92  * corresponding anonymous type name.
93  * </li>
94  * </ol>
95  * </li>
96  * <li> If the <b>[member type definition anonymous]</b> exists:
97  * <ol>
98  * <li>If it is false, then expose <b>[member type definition name]</b> and <b>[member type
    definition namespace]</b> properties;
99  * </li>
100 * <li>Otherwise, expose the namespace and local name of the
101 * corresponding anonymous type name.
102 * </li>
103 * </ol>
104 * </li>
105 * <li> If the <b>[type definition anonymous]</b> exists:
106 * <ol>
107 * <li>If it is false, then expose <b>[type definition name]</b> and <b>[type definition namespace
    ]</b> properties;
108 * </li>
109 * <li>Otherwise, expose the namespace and local name of the
110 * corresponding anonymous type name.
111 * </li>
112 * </ol>
113 * </li>
114 * </ol>
115 * </li>
116 * </ul>
117 * <p><b>Note:</b> Other schema languages are outside the scope of the W3C
118 * and therefore should define how to represent their type systems using
119 * <code>TypeInfo</code>.
120 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>Document
    Object Model (DOM) Level 3 Core Specification</a>.
121 * @since DOM Level 3
122 */
123 public interface TypeInfo {
124     /**
125      * The name of a type declared for the associated element or attribute,
126      * or <code>null</code> if unknown.
127      */
128     public String getTypeName();
129
130     /**
131      * The namespace of the type declared for the associated element or
132      * attribute or <code>null</code> if the element does not have

```

```

133     * declaration or if no namespace information is available.
134     */
135     public String getTypeNamespace();
136
137     // DerivationMethods
138     /**
139     * If the document's schema is an XML Schema [
```

```

-1-20010502/'>XML Schema Part 1</a>]
176 * , this constant represents the <a href='http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#
    element-list'>list</a>.
177 * <br> The reference type definition is derived by list from the other
178 * type definition if there exists two type definitions T1 and T2 such
179 * as the reference type definition is derived from T1 by
180 * <code>DERIVATION_RESTRICTION</code> or
181 * <code>DERIVATION_EXTENSION</code>, T2 is derived from the other type
182 * definition by <code>DERIVATION_RESTRICTION</code>, T1 has {variety} <em>list</em>, and T2 is
    the {item type definition}. Note that T1 could be the same as
183 * the reference type definition, and T2 could be the same as the other
184 * type definition.
185 */
186 public static final int DERIVATION_LIST          = 0x00000008;
187
188 /**
189 * This method returns if there is a derivation between the reference
190 * type definition, i.e. the <code>TypeInfo</code> on which the method
191 * is being called, and the other type definition, i.e. the one passed
192 * as parameters.
193 * @param typeNamespaceArg the namespace of the other type definition.
194 * @param typeNameArg the name of the other type definition.
195 * @param derivationMethod the type of derivation and conditions applied
196 * between two types, as described in the list of constants provided
197 * in this interface.
198 * @return If the document's schema is a DTD or no schema is associated
199 * with the document, this method will always return <code>false</code>
200 * . If the document's schema is an XML Schema, the method will return
201 * <code>true</code> if the reference type definition is derived from
202 * the other type definition according to the derivation parameter. If
203 * the value of the parameter is <code>0</code> (no bit is set to
204 * <code>1</code> for the <code>derivationMethod</code> parameter),
205 * the method will return <code>true</code> if the other type
206 * definition can be reached by recursing any combination of {base
207 * type definition}, {item type definition}, or {member type
208 * definitions} from the reference type definition.
209 */
210 public boolean isDerivedFrom(String typeNamespaceArg,
211                             String typeNameArg,
212                             int derivationMethod);
213
214 }

```

## 36.28 UserDataHandler.java

Secuencia 359: org/w3c/dom/UserDataHandler.java

```

1 /*
2 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3 *
4 *
5 *
6 *
7 *

```

```

 8  *
 9  *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25 /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42 package org.w3c.dom;
43
44 /**
45  * When associating an object to a key on a node using
46  * Node.setUserData() the application can provide a handler
47  * that gets called when the node the object is associated to is being
48  * cloned, imported, or renamed. This can be used by the application to
49  * implement various behaviors regarding the data it associates to the DOM
50  * nodes. This interface defines that handler.
51  * 

See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 Document
52  * Object Model (DOM) Level 3 Core Specification.


53  * @since DOM Level 3
54  */
55 public interface UserDataHandler {
56     // OperationType
57     /**
58      * The node is cloned, using Node.cloneNode().
59      */
60     public static final short NODE_CLONED = 1;

```

```

60  /**
61   * The node is imported, using Document.importNode().
62   */
63  public static final short NODE_IMPORTED          = 2;
64  /**
65   * The node is deleted.
66   * 

><b>Note:</b> This may not be supported or may not be reliable in
67   * certain environments, such as Java, where the implementation has no
68   * real control over when objects are actually deleted.
69   */
70  public static final short NODE_DELETED           = 3;
71  /**
72   * The node is renamed, using Document.renameNode().
73   */
74  public static final short NODE_RENAMED           = 4;
75  /**
76   * The node is adopted, using Document.adoptNode().
77   */
78  public static final short NODE_ADOPTED           = 5;
79
80  /**
81   * This method is called whenever the node for which this handler is
82   * registered is imported or cloned.
83   *   
 DOM applications must not raise exceptions in a
84   * UserDataHandler. The effect of throwing exceptions from
85   * the handler is DOM implementation dependent.
86   * @param operation Specifies the type of operation that is being
87   *   performed on the node.
88   * @param key Specifies the key for which this handler is being called.
89   * @param data Specifies the data for which this handler is being called.
90   * @param src Specifies the node being cloned, adopted, imported, or
91   *   renamed. This is null when the node is being deleted.
92   * @param dst Specifies the node newly created if any, or
93   *   null.
94   */
95  public void handle(short operation,
96                     String key,
97                     Object data,
98                     Node src,
99                     Node dst);
100
101 }


```

## 37 Dom Bootstrap (org.w3c.dom.bootstrap package)

### 37.1 DOMImplementationRegistry.java

Secuencia 360: org/w3c/dom/bootstrap/DOMImplementationRegistry.java

```

1  /**
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *

```

```
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42
43 package org.w3c.dom.bootstrap;
44
45 import java.util.StringTokenizer;
46 import java.util.Vector;
47 import org.w3c.dom.DOMImplementationSource;
48 import org.w3c.dom.DOMImplementationList;
49 import org.w3c.dom.DOMImplementation;
50 import java.io.InputStream;
51 import java.io.BufferedReader;
52 import java.io.InputStreamReader;
53 import java.security.AccessController;
54 import java.security.PrivilegedAction;
55
56 /**
57  * A factory that enables applications to obtain instances of
58  * DOMImplementation.
```



```

59  *
60  * <p>
61  * Example:
62  * </p>
63  *
64  * <pre class='example'>
65  * // get an instance of the DOMImplementation registry
66  * DOMImplementationRegistry registry =
67  *     DOMImplementationRegistry.newInstance();
68  * // get a DOM implementation the Level 3 XML module
69  * DOMImplementation domImpl =
70  *     registry.getDOMImplementation("XML 3.0");
71  * </pre>
72  *
73  * <p>
74  * This provides an application with an implementation-independent starting
75  * point. DOM implementations may modify this class to meet new security
76  * standards or to provide *additional* fallbacks for the list of
77  * DOMImplementationSources.
78  * </p>
79  *
80  * @see DOMImplementation
81  * @see DOMImplementationSource
82  * @since DOM Level 3
83  */
84  public final class DOMImplementationRegistry {
85      /**
86       * The system property to specify the
87       * DOMImplementationSource class names.
88       */
89      public static final String PROPERTY =
90          "org.w3c.dom.DOMImplementationSourceList";
91
92      /**
93       * Default columns per line.
94       */
95      private static final int DEFAULT_LINE_LENGTH = 80;
96
97      /**
98       * The list of DOMImplementationSources.
99       */
100     private Vector sources;
101
102     /**
103      * Default class name.
104      */
105     private static final String FALLBACK_CLASS =
106         "com.sun.org.apache.xerces.internal.dom.DOMXSImplementationSourceImpl";
107     private static final String DEFAULT_PACKAGE =
108         "com.sun.org.apache.xerces.internal.dom";
109
110     /**
111      * Private constructor.
112      * @param srcs Vector List of DOMImplementationSources

```

```

112     */
113     private DOMImplementationRegistry(final Vector srcs) {
114         sources = srcs;
115     }
116
117     /**
118     * Obtain a new instance of a DOMImplementationRegistry.
119     *
120     * The DOMImplementationRegistry is initialized by the
121     * application or the implementation, depending on the context, by
122     * first checking the value of the Java system property
123     * org.w3c.dom.DOMImplementationSourceList and
124     * the service provider whose contents are at
125     * "META-INF/services/org.w3c.dom.DOMImplementationSourceList".
126     * The value of this property is a white-space separated list of
127     * names of available classes implementing the
128     * DOMImplementationSource interface. Each class listed
129     * in the class name list is instantiated and any exceptions
130     * encountered are thrown to the application.
131     *
132     * @return an initialized instance of DOMImplementationRegistry
133     * @throws ClassNotFoundException
134     *         If any specified class can not be found
135     * @throws InstantiationException
136     *         If any specified class is an interface or abstract class
137     * @throws IllegalAccessException
138     *         If the default constructor of a specified class is not accessible
139     * @throws ClassCastException
140     *         If any specified class does not implement
141     *         DOMImplementationSource
142     */
143     public static DOMImplementationRegistry newInstance()
144         throws
145         ClassNotFoundException,
146         InstantiationException,
147         IllegalAccessException,
148         ClassCastException {
149         Vector sources = new Vector();
150
151         ClassLoader classLoader = getClassLoader();
152         // fetch system property:
153         String p = getSystemProperty(PROPERTY);
154
155         //
156         // if property is not specified then use contents of
157         // META-INF/org.w3c.dom.DOMImplementationSourceList from classpath
158         if (p == null) {
159             p = getServiceValue(classLoader);
160         }
161         if (p == null) {
162             //
163             // DOM Implementations can modify here to add *additional* fallback

```

```

165         // mechanisms to access a list of default DOMImplementationSources.
166         //fall back to JAXP implementation class com.sun.org.apache.xerces.internal.dom.
167         DOMXSImplementationSourceImpl
168         p = FALLBACK_CLASS;
169     }
170     if (p != null) {
171         StringTokenizer st = new StringTokenizer(p);
172         while (st.hasMoreTokens()) {
173             String sourceName = st.nextToken();
174             // make sure we have access to restricted packages
175             boolean internal = false;
176             if (System.getSecurityManager() != null) {
177                 if (sourceName != null && sourceName.startsWith(DEFAULT_PACKAGE)) {
178                     internal = true;
179                 }
180             }
181             Class sourceClass = null;
182             if (classLoader != null && !internal) {
183                 sourceClass = classLoader.loadClass(sourceName);
184             } else {
185                 sourceClass = Class.forName(sourceName);
186             }
187             DOMImplementationSource source =
188                 (DOMImplementationSource) sourceClass.newInstance();
189             sources.addElement(source);
190         }
191     }
192     return new DOMImplementationRegistry(sources);
193 }
194 /**
195  * Return the first implementation that has the desired
196  * features, or <code>null</code> if none is found.
197  *
198  * @param features
199  *      A string that specifies which features are required. This is
200  *      a space separated list in which each feature is specified by
201  *      its name optionally followed by a space and a version number.
202  *      This is something like: "XML 1.0 Traversal +Events 2.0"
203  * @return An implementation that has the desired features,
204  *      or <code>null</code> if none found.
205  */
206 public DOMImplementation getDOMImplementation(final String features) {
207     int size = sources.size();
208     String name = null;
209     for (int i = 0; i < size; i++) {
210         DOMImplementationSource source =
211             (DOMImplementationSource) sources.elementAt(i);
212         DOMImplementation impl = source.getDOMImplementation(features);
213         if (impl != null) {
214             return impl;
215         }
216     }

```

```

217         return null;
218     }
219
220     /**
221      * Return a list of implementations that support the
222      * desired features.
223      *
224      * @param features
225      *      A string that specifies which features are required. This is
226      *      a space separated list in which each feature is specified by
227      *      its name optionally followed by a space and a version number.
228      *      This is something like: "XML 1.0 Traversal +Events 2.0"
229      * @return A list of DOMImplementations that support the desired features.
230      */
231     public DOMImplementationList getDOMImplementationList(final String features) {
232         final Vector implementations = new Vector();
233         int size = sources.size();
234         for (int i = 0; i < size; i++) {
235             DOMImplementationSource source =
236                 (DOMImplementationSource) sources.elementAt(i);
237             DOMImplementationList impls =
238                 source.getDOMImplementationList(features);
239             for (int j = 0; j < impls.getLength(); j++) {
240                 DOMImplementation impl = impls.item(j);
241                 implementations.addElement(impl);
242             }
243         }
244         return new DOMImplementationList() {
245             public DOMImplementation item(final int index) {
246                 if (index >= 0 && index < implementations.size()) {
247                     try {
248                         return (DOMImplementation)
249                             implementations.elementAt(index);
250                     } catch (ArrayIndexOutOfBoundsException e) {
251                         return null;
252                     }
253                 }
254                 return null;
255             }
256
257             public int getLength() {
258                 return implementations.size();
259             }
260         };
261     }
262
263     /**
264      * Register an implementation.
265      *
266      * @param s The source to be registered, may not be <code>null</code>
267      */
268     public void addSource(final DOMImplementationSource s) {
269         if (s == null) {

```

```

270         throw new NullPointerException();
271     }
272     if (!sources.contains(s)) {
273         sources.addElement(s);
274     }
275 }
276
277 /**
278  *
279  * Gets a class loader.
280  *
281  * @return A class loader, possibly null
282  */
283 private static ClassLoader getClassLoader() {
284     try {
285         ClassLoader contextClassLoader = getContextClassLoader();
286
287         if (contextClassLoader != null) {
288             return contextClassLoader;
289         }
290     } catch (Exception e) {
291         // Assume that the DOM application is in a JRE 1.1, use the
292         // current ClassLoader
293         return DOMImplementationRegistry.class.getClassLoader();
294     }
295     return DOMImplementationRegistry.class.getClassLoader();
296 }
297
298 /**
299  * This method attempts to return the first line of the resource
300  * META-INF/services/org.w3c.dom.DOMImplementationSourceList
301  * from the provided ClassLoader.
302  *
303  * @param classLoader classLoader, may not be null.
304  * @return first line of resource, or null
305  */
306 private static String getServiceValue(final ClassLoader classLoader) {
307     String serviceId = "META-INF/services/" + PROPERTY;
308     // try to find services in CLASSPATH
309     try {
310         InputStream is = getResourceAsStream(classLoader, serviceId);
311
312         if (is != null) {
313             BufferedReader rd;
314             try {
315                 rd =
316                     new BufferedReader(new InputStreamReader(is, "UTF-8"),
317                                     DEFAULT_LINE_LENGTH);
318             } catch (java.io.UnsupportedEncodingException e) {
319                 rd =
320                     new BufferedReader(new InputStreamReader(is),
321                                     DEFAULT_LINE_LENGTH);
322             }

```

```

323         String serviceValue = rd.readLine();
324         rd.close();
325         if (serviceValue != null && serviceValue.length() > 0) {
326             return serviceValue;
327         }
328     }
329 } catch (Exception ex) {
330     return null;
331 }
332 return null;
333 }
334
335 /**
336  * A simple JRE (Java Runtime Environment) 1.1 test
337  *
338  * @return <code>true</code> if JRE 1.1
339  */
340 private static boolean isJRE11() {
341     try {
342         Class c = Class.forName("java.security.AccessController");
343         // java.security.AccessController existed since 1.2 so, if no
344         // exception was thrown, the DOM application is running in a JRE
345         // 1.2 or higher
346         return false;
347     } catch (Exception ex) {
348         // ignore
349     }
350     return true;
351 }
352
353 /**
354  * This method returns the ContextClassLoader or <code>null</code> if
355  * running in a JRE 1.1
356  *
357  * @return The Context Classloader
358  */
359 private static ClassLoader getContextClassLoader() {
360     return isJRE11()
361         ? null
362         : (ClassLoader)
363           AccessController.doPrivileged(new PrivilegedAction() {
364               public Object run() {
365                   ClassLoader classLoader = null;
366                   try {
367                       classLoader =
368                           Thread.currentThread().getContextClassLoader();
369                   } catch (SecurityException ex) {
370                       }
371                   return classLoader;
372               }
373           });
374 }
375

```

```
376 /**
377  * This method returns the system property indicated by the specified name
378  * after checking access control privileges. For a JRE 1.1, this check is
379  * not done.
380  *
381  * @param name the name of the system property
382  * @return the system property
383  */
384 private static String getSystemProperty(final String name) {
385     return isJRE11()
386         ? (String) System.getProperty(name)
387         : (String) AccessController.doPrivileged(new PrivilegedAction() {
388             public Object run() {
389                 return System.getProperty(name);
390             }
391         });
392 }
393
394 /**
395  * This method returns an InputStream for the reading resource
396  * META-INF/services/org.w3c.dom.DOMImplementationSourceList after checking
397  * access control privileges. For a JRE 1.1, this check is not done.
398  *
399  * @param classLoader classLoader
400  * @param name the resource
401  * @return an InputStream for the resource specified
402  */
403 private static InputStream getResourceAsStream(final ClassLoader classLoader,
404                                               final String name) {
405     if (isJRE11()) {
406         InputStream ris;
407         if (classLoader == null) {
408             ris = ClassLoader.getSystemResourceAsStream(name);
409         } else {
410             ris = classLoader.getResourceAsStream(name);
411         }
412         return ris;
413     } else {
414         return (InputStream)
415             AccessController.doPrivileged(new PrivilegedAction() {
416                 public Object run() {
417                     InputStream ris;
418                     if (classLoader == null) {
419                         ris =
420                             ClassLoader.getSystemResourceAsStream(name);
421                     } else {
422                         ris = classLoader.getResourceAsStream(name);
423                     }
424                     return ris;
425                 }
426             });
427     }
428 }
```

429 }

## 38 Dom Css (org.w3c.dom.css package)

### 38.1 CSS2Properties.java

Secuencia 361: org/w3c/dom/css/CSS2Properties.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.DOMException;
45
46 /**
```



```

47 * The <code>CSS2Properties</code> interface represents a convenience
48 * mechanism for retrieving and setting properties within a
49 * <code>CSSStyleDeclaration</code>. The attributes of this interface
50 * correspond to all the properties specified in CSS2. Getting an attribute
51 * of this interface is equivalent to calling the
52 * <code>getPropertyValue</code> method of the
53 * <code>CSSStyleDeclaration</code> interface. Setting an attribute of this
54 * interface is equivalent to calling the <code>setProperty</code> method of
55 * the <code>CSSStyleDeclaration</code> interface.
56 * <p> A conformant implementation of the CSS module is not required to
57 * implement the <code>CSS2Properties</code> interface. If an implementation
58 * does implement this interface, the expectation is that language-specific
59 * methods can be used to cast from an instance of the
60 * <code>CSSStyleDeclaration</code> interface to the
61 * <code>CSS2Properties</code> interface.
62 * <p> If an implementation does implement this interface, it is expected to
63 * understand the specific syntax of the shorthand properties, and apply
64 * their semantics; when the <code>margin</code> property is set, for
65 * example, the <code>marginTop</code>, <code>marginRight</code>,
66 * <code>marginBottom</code> and <code>marginLeft</code> properties are
67 * actually being set by the underlying implementation.
68 * <p> When dealing with CSS "shorthand" properties, the shorthand properties
69 * should be decomposed into their component longhand properties as
70 * appropriate, and when querying for their value, the form returned should
71 * be the shortest form exactly equivalent to the declarations made in the
72 * ruleset. However, if there is no shorthand declaration that could be
73 * added to the ruleset without changing in any way the rules already
74 * declared in the ruleset (i.e., by adding longhand rules that were
75 * previously not declared in the ruleset), then the empty string should be
76 * returned for the shorthand property.
77 * <p> For example, querying for the <code>font</code> property should not
78 * return "normal normal normal 14pt/normal Arial, sans-serif", when "14pt
79 * Arial, sans-serif" suffices. (The normals are initial values, and are
80 * implied by use of the longhand property.)
81 * <p> If the values for all the longhand properties that compose a particular
82 * string are the initial values, then a string consisting of all the
83 * initial values should be returned (e.g. a <code>border-width</code> value
84 * of "medium" should be returned as such, not as "").
85 * <p> For some shorthand properties that take missing values from other
86 * sides, such as the <code>margin</code>, <code>padding</code>, and
87 * <code>border-[width|style|color]</code> properties, the minimum number of
88 * sides possible should be used; i.e., "0px 10px" will be returned instead
89 * of "0px 10px 0px 10px".
90 * <p> If the value of a shorthand property can not be decomposed into its
91 * component longhand properties, as is the case for the <code>font</code>
92 * property with a value of "menu", querying for the values of the component
93 * longhand properties should return the empty string.
94 * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
    Object Model (DOM) Level 2 Style Specification</a>.
95 * @since DOM Level 2
96 */
97 public interface CSS2Properties {
98     /**

```

```
99      * See the azimuth property definition in CSS2.
100     */
101     public String getAzimuth();
102     /**
103      * See the azimuth property definition in CSS2.
104      * @exception DOMException
105      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
106      *     unparsable.
107      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
108      */
109     public void setAzimuth(String azimuth)
110         throws DOMException;
111
112     /**
113      * See the background property definition in CSS2.
114      */
115     public String getBackground();
116     /**
117      * See the background property definition in CSS2.
118      * @exception DOMException
119      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
120      *     unparsable.
121      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
122      */
123     public void setBackground(String background)
124         throws DOMException;
125
126     /**
127      * See the background-attachment property definition in CSS2.
128      */
129     public String getBackgroundAttachment();
130     /**
131      * See the background-attachment property definition in CSS2.
132      * @exception DOMException
133      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
134      *     unparsable.
135      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
136      */
137     public void setBackgroundAttachment(String backgroundAttachment)
138         throws DOMException;
139
140     /**
141      * See the background-color property definition in CSS2.
142      */
143     public String getBackgroundColor();
144     /**
145      * See the background-color property definition in CSS2.
146      * @exception DOMException
147      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
148      *     unparsable.
149      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
150      */
151     public void setBackgroundColor(String backgroundColor)
```

```

152         throws DOMException;
153
154     /**
155     * See the background-image property definition in CSS2.
156     */
157     public String getBackgroundImage();
158     /**
159     * See the background-image property definition in CSS2.
160     * @exception DOMException
161     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
162     *     unparsable.
163     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
164     */
165     public void setBackgroundImage(String backgroundImage)
166         throws DOMException;
167
168     /**
169     * See the background-position property definition in CSS2.
170     */
171     public String getBackgroundPosition();
172     /**
173     * See the background-position property definition in CSS2.
174     * @exception DOMException
175     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
176     *     unparsable.
177     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
178     */
179     public void setBackgroundPosition(String backgroundPosition)
180         throws DOMException;
181
182     /**
183     * See the background-repeat property definition in CSS2.
184     */
185     public String getBackgroundRepeat();
186     /**
187     * See the background-repeat property definition in CSS2.
188     * @exception DOMException
189     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
190     *     unparsable.
191     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
192     */
193     public void setBackgroundRepeat(String backgroundRepeat)
194         throws DOMException;
195
196     /**
197     * See the border property definition in CSS2.
198     */
199     public String getBorder();
200     /**
201     * See the border property definition in CSS2.
202     * @exception DOMException
203     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
204     *     unparsable.

```

```
205     * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
206     */
207     public void setBorder(String border)
208                             throws DOMException;
209
210     /**
211     * See the border-collapse property definition in CSS2.
212     */
213     public String getBorderCollapse();
214     /**
215     * See the border-collapse property definition in CSS2.
216     * @exception DOMException
217     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
218     *     unparsable.
219     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
220     */
221     public void setBorderCollapse(String borderCollapse)
222                             throws DOMException;
223
224     /**
225     * See the border-color property definition in CSS2.
226     */
227     public String getBorderColor();
228     /**
229     * See the border-color property definition in CSS2.
230     * @exception DOMException
231     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
232     *     unparsable.
233     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
234     */
235     public void setBorderColor(String borderColor)
236                             throws DOMException;
237
238     /**
239     * See the border-spacing property definition in CSS2.
240     */
241     public String getBorderSpacing();
242     /**
243     * See the border-spacing property definition in CSS2.
244     * @exception DOMException
245     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
246     *     unparsable.
247     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
248     */
249     public void setBorderSpacing(String borderSpacing)
250                             throws DOMException;
251
252     /**
253     * See the border-style property definition in CSS2.
254     */
255     public String getBorderStyle();
256     /**
257     * See the border-style property definition in CSS2.
```

```
258     * @exception DOMException
259     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
260     *     unparsable.
261     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
262     */
263     public void setBorderStyle(String borderStyle)
264         throws DOMException;
265
266     /**
267     * See the border-top property definition in CSS2.
268     */
269     public String getBorderTop();
270     /**
271     * See the border-top property definition in CSS2.
272     * @exception DOMException
273     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
274     *     unparsable.
275     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
276     */
277     public void setBorderTop(String borderTop)
278         throws DOMException;
279
280     /**
281     * See the border-right property definition in CSS2.
282     */
283     public String getBorderRight();
284     /**
285     * See the border-right property definition in CSS2.
286     * @exception DOMException
287     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
288     *     unparsable.
289     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
290     */
291     public void setBorderRight(String borderRight)
292         throws DOMException;
293
294     /**
295     * See the border-bottom property definition in CSS2.
296     */
297     public String getBorderBottom();
298     /**
299     * See the border-bottom property definition in CSS2.
300     * @exception DOMException
301     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
302     *     unparsable.
303     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
304     */
305     public void setBorderBottom(String borderBottom)
306         throws DOMException;
307
308     /**
309     * See the border-left property definition in CSS2.
310     */
```

```
311 public String getBorderLeft();
312 /**
313  * See the border-left property definition in CSS2.
314  * @exception DOMException
315  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
316  *   unparsable.
317  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
318  */
319 public void setBorderLeft(String borderLeft)
320     throws DOMException;
321
322 /**
323  * See the border-top-color property definition in CSS2.
324  */
325 public String getBorderTopColor();
326 /**
327  * See the border-top-color property definition in CSS2.
328  * @exception DOMException
329  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
330  *   unparsable.
331  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
332  */
333 public void setBorderTopColor(String borderTopColor)
334     throws DOMException;
335
336 /**
337  * See the border-right-color property definition in CSS2.
338  */
339 public String getBorderRightColor();
340 /**
341  * See the border-right-color property definition in CSS2.
342  * @exception DOMException
343  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
344  *   unparsable.
345  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
346  */
347 public void setBorderRightColor(String borderRightColor)
348     throws DOMException;
349
350 /**
351  * See the border-bottom-color property definition in CSS2.
352  */
353 public String getBorderBottomColor();
354 /**
355  * See the border-bottom-color property definition in CSS2.
356  * @exception DOMException
357  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
358  *   unparsable.
359  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
360  */
361 public void setBorderBottomColor(String borderBottomColor)
362     throws DOMException;
363
```

```
364 /**
365  * See the border-left-color property definition in CSS2.
366  */
367 public String getBorderLeftColor();
368 /**
369  * See the border-left-color property definition in CSS2.
370  * @exception DOMException
371  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
372  *   unparsable.
373  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
374  */
375 public void setBorderLeftColor(String borderLeftColor)
376     throws DOMException;
377
378 /**
379  * See the border-top-style property definition in CSS2.
380  */
381 public String getBorderTopStyle();
382 /**
383  * See the border-top-style property definition in CSS2.
384  * @exception DOMException
385  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
386  *   unparsable.
387  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
388  */
389 public void setBorderTopStyle(String borderTopStyle)
390     throws DOMException;
391
392 /**
393  * See the border-right-style property definition in CSS2.
394  */
395 public String getBorderRightStyle();
396 /**
397  * See the border-right-style property definition in CSS2.
398  * @exception DOMException
399  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
400  *   unparsable.
401  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
402  */
403 public void setBorderRightStyle(String borderRightStyle)
404     throws DOMException;
405
406 /**
407  * See the border-bottom-style property definition in CSS2.
408  */
409 public String getBorderBottomStyle();
410 /**
411  * See the border-bottom-style property definition in CSS2.
412  * @exception DOMException
413  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
414  *   unparsable.
415  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
416  */
```

```
417 public void setBorderBottomStyle(String borderBottomStyle)
418     throws DOMException;
419
420 /**
421  * See the border-left-style property definition in CSS2.
422  */
423 public String getBorderLeftStyle();
424 /**
425  * See the border-left-style property definition in CSS2.
426  * @exception DOMException
427  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
428  *   unparsable.
429  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
430  */
431 public void setBorderLeftStyle(String borderLeftStyle)
432     throws DOMException;
433
434 /**
435  * See the border-top-width property definition in CSS2.
436  */
437 public String getBorderTopWidth();
438 /**
439  * See the border-top-width property definition in CSS2.
440  * @exception DOMException
441  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
442  *   unparsable.
443  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
444  */
445 public void setBorderTopWidth(String borderTopWidth)
446     throws DOMException;
447
448 /**
449  * See the border-right-width property definition in CSS2.
450  */
451 public String getBorderRightWidth();
452 /**
453  * See the border-right-width property definition in CSS2.
454  * @exception DOMException
455  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
456  *   unparsable.
457  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
458  */
459 public void setBorderRightWidth(String borderRightWidth)
460     throws DOMException;
461
462 /**
463  * See the border-bottom-width property definition in CSS2.
464  */
465 public String getBorderBottomWidth();
466 /**
467  * See the border-bottom-width property definition in CSS2.
468  * @exception DOMException
469  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
```



```
470     * unparsable.
471     * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
472     */
473     public void setBorderBottomWidth(String borderBottomWidth)
474         throws DOMException;
475
476     /**
477     * See the border-left-width property definition in CSS2.
478     */
479     public String getBorderLeftWidth();
480     /**
481     * See the border-left-width property definition in CSS2.
482     * @exception DOMException
483     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
484     *     unparsable.
485     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
486     */
487     public void setBorderLeftWidth(String borderLeftWidth)
488         throws DOMException;
489
490     /**
491     * See the border-width property definition in CSS2.
492     */
493     public String getBorderWidth();
494     /**
495     * See the border-width property definition in CSS2.
496     * @exception DOMException
497     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
498     *     unparsable.
499     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
500     */
501     public void setBorderWidth(String borderWidth)
502         throws DOMException;
503
504     /**
505     * See the bottom property definition in CSS2.
506     */
507     public String getBottom();
508     /**
509     * See the bottom property definition in CSS2.
510     * @exception DOMException
511     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
512     *     unparsable.
513     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
514     */
515     public void setBottom(String bottom)
516         throws DOMException;
517
518     /**
519     * See the caption-side property definition in CSS2.
520     */
521     public String getCaptionSide();
522     /**
```

```
523     * See the caption-side property definition in CSS2.
524     * @exception DOMException
525     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
526     *     unparsable.
527     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
528     */
529     public void setCaptionSide(String captionSide)
530         throws DOMException;
531
532     /**
533     * See the clear property definition in CSS2.
534     */
535     public String getClear();
536     /**
537     * See the clear property definition in CSS2.
538     * @exception DOMException
539     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
540     *     unparsable.
541     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
542     */
543     public void setClear(String clear)
544         throws DOMException;
545
546     /**
547     * See the clip property definition in CSS2.
548     */
549     public String getClip();
550     /**
551     * See the clip property definition in CSS2.
552     * @exception DOMException
553     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
554     *     unparsable.
555     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
556     */
557     public void setClip(String clip)
558         throws DOMException;
559
560     /**
561     * See the color property definition in CSS2.
562     */
563     public String getColor();
564     /**
565     * See the color property definition in CSS2.
566     * @exception DOMException
567     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
568     *     unparsable.
569     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
570     */
571     public void setColor(String color)
572         throws DOMException;
573
574     /**
575     * See the content property definition in CSS2.
```

```
576     */
577     public String getContent();
578     /**
579     * See the content property definition in CSS2.
580     * @exception DOMException
581     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
582     *     unparsable.
583     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
584     */
585     public void setContent(String content)
586                             throws DOMException;
587
588     /**
589     * See the counter-increment property definition in CSS2.
590     */
591     public String getCounterIncrement();
592     /**
593     * See the counter-increment property definition in CSS2.
594     * @exception DOMException
595     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
596     *     unparsable.
597     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
598     */
599     public void setCounterIncrement(String counterIncrement)
600                                     throws DOMException;
601
602     /**
603     * See the counter-reset property definition in CSS2.
604     */
605     public String getCounterReset();
606     /**
607     * See the counter-reset property definition in CSS2.
608     * @exception DOMException
609     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
610     *     unparsable.
611     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
612     */
613     public void setCounterReset(String counterReset)
614                                     throws DOMException;
615
616     /**
617     * See the cue property definition in CSS2.
618     */
619     public String getCue();
620     /**
621     * See the cue property definition in CSS2.
622     * @exception DOMException
623     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
624     *     unparsable.
625     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
626     */
627     public void setCue(String cue)
628                             throws DOMException;
```

```
629
630 /**
631  * See the cue-after property definition in CSS2.
632  */
633 public String getCueAfter();
634 /**
635  * See the cue-after property definition in CSS2.
636  * @exception DOMException
637  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
638  *   unparsable.
639  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
640  */
641 public void setCueAfter(String cueAfter)
642                               throws DOMException;
643
644 /**
645  * See the cue-before property definition in CSS2.
646  */
647 public String getCueBefore();
648 /**
649  * See the cue-before property definition in CSS2.
650  * @exception DOMException
651  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
652  *   unparsable.
653  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
654  */
655 public void setCueBefore(String cueBefore)
656                               throws DOMException;
657
658 /**
659  * See the cursor property definition in CSS2.
660  */
661 public String getCursor();
662 /**
663  * See the cursor property definition in CSS2.
664  * @exception DOMException
665  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
666  *   unparsable.
667  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
668  */
669 public void setCursor(String cursor)
670                               throws DOMException;
671
672 /**
673  * See the direction property definition in CSS2.
674  */
675 public String getDirection();
676 /**
677  * See the direction property definition in CSS2.
678  * @exception DOMException
679  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
680  *   unparsable.
681  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
```

```
682     */
683     public void setDirection(String direction)
684         throws DOMException;
685
686     /**
687     * See the display property definition in CSS2.
688     */
689     public String getDisplay();
690     /**
691     * See the display property definition in CSS2.
692     * @exception DOMException
693     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
694     *     unparsable.
695     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
696     */
697     public void setDisplay(String display)
698         throws DOMException;
699
700     /**
701     * See the elevation property definition in CSS2.
702     */
703     public String getElevation();
704     /**
705     * See the elevation property definition in CSS2.
706     * @exception DOMException
707     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
708     *     unparsable.
709     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
710     */
711     public void setElevation(String elevation)
712         throws DOMException;
713
714     /**
715     * See the empty-cells property definition in CSS2.
716     */
717     public String getEmptyCells();
718     /**
719     * See the empty-cells property definition in CSS2.
720     * @exception DOMException
721     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
722     *     unparsable.
723     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
724     */
725     public void setEmptyCells(String emptyCells)
726         throws DOMException;
727
728     /**
729     * See the float property definition in CSS2.
730     */
731     public String getCssFloat();
732     /**
733     * See the float property definition in CSS2.
734     * @exception DOMException
```

```
735 * SYNTAX_ERR: Raised if the new value has a syntax error and is
736 * unparsable.
737 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
738 */
739 public void setCssFloat(String cssFloat)
740     throws DOMException;
741
742 /**
743 * See the font property definition in CSS2.
744 */
745 public String getFont();
746 /**
747 * See the font property definition in CSS2.
748 * @exception DOMException
749 * SYNTAX_ERR: Raised if the new value has a syntax error and is
750 * unparsable.
751 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
752 */
753 public void setFont(String font)
754     throws DOMException;
755
756 /**
757 * See the font-family property definition in CSS2.
758 */
759 public String getFontFamily();
760 /**
761 * See the font-family property definition in CSS2.
762 * @exception DOMException
763 * SYNTAX_ERR: Raised if the new value has a syntax error and is
764 * unparsable.
765 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
766 */
767 public void setFontFamily(String fontFamily)
768     throws DOMException;
769
770 /**
771 * See the font-size property definition in CSS2.
772 */
773 public String getFontSize();
774 /**
775 * See the font-size property definition in CSS2.
776 * @exception DOMException
777 * SYNTAX_ERR: Raised if the new value has a syntax error and is
778 * unparsable.
779 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
780 */
781 public void setFontSize(String fontSize)
782     throws DOMException;
783
784 /**
785 * See the font-size-adjust property definition in CSS2.
786 */
787 public String getFontSizeAdjust();
```

```
788 /**
789  * See the font-size-adjust property definition in CSS2.
790  * @exception DOMException
791  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
792  *   unparsable.
793  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
794  */
795 public void setFontSizeAdjust(String fontSizeAdjust)
796     throws DOMException;
797
798 /**
799  * See the font-stretch property definition in CSS2.
800  */
801 public String getFontStretch();
802 /**
803  * See the font-stretch property definition in CSS2.
804  * @exception DOMException
805  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
806  *   unparsable.
807  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
808  */
809 public void setFontStretch(String fontStretch)
810     throws DOMException;
811
812 /**
813  * See the font-style property definition in CSS2.
814  */
815 public String getFontStyle();
816 /**
817  * See the font-style property definition in CSS2.
818  * @exception DOMException
819  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
820  *   unparsable.
821  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
822  */
823 public void setFontStyle(String fontStyle)
824     throws DOMException;
825
826 /**
827  * See the font-variant property definition in CSS2.
828  */
829 public String getFontVariant();
830 /**
831  * See the font-variant property definition in CSS2.
832  * @exception DOMException
833  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
834  *   unparsable.
835  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
836  */
837 public void setFontVariant(String fontVariant)
838     throws DOMException;
839
840 /**
```

```
841     * See the font-weight property definition in CSS2.
842     */
843     public String getFontWeight();
844     /**
845     * See the font-weight property definition in CSS2.
846     * @exception DOMException
847     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
848     *     unparsable.
849     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
850     */
851     public void setFontWeight(String fontWeight)
852         throws DOMException;
853
854     /**
855     * See the height property definition in CSS2.
856     */
857     public String getHeight();
858     /**
859     * See the height property definition in CSS2.
860     * @exception DOMException
861     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
862     *     unparsable.
863     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
864     */
865     public void setHeight(String height)
866         throws DOMException;
867
868     /**
869     * See the left property definition in CSS2.
870     */
871     public String getLeft();
872     /**
873     * See the left property definition in CSS2.
874     * @exception DOMException
875     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
876     *     unparsable.
877     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
878     */
879     public void setLeft(String left)
880         throws DOMException;
881
882     /**
883     * See the letter-spacing property definition in CSS2.
884     */
885     public String getLetterSpacing();
886     /**
887     * See the letter-spacing property definition in CSS2.
888     * @exception DOMException
889     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
890     *     unparsable.
891     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
892     */
893     public void setLetterSpacing(String letterSpacing)
```



```
894         throws DOMException;
895
896     /**
897     * See the line-height property definition in CSS2.
898     */
899     public String getLineHeight();
900
901     /**
902     * See the line-height property definition in CSS2.
903     * @exception DOMException
904     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
905     *     unparsable.
906     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
907     */
908     public void setLineHeight(String lineHeight)
909         throws DOMException;
910
911     /**
912     * See the list-style property definition in CSS2.
913     */
914     public String getListStyle();
915
916     /**
917     * See the list-style property definition in CSS2.
918     * @exception DOMException
919     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
920     *     unparsable.
921     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
922     */
923     public void setListStyle(String listStyle)
924         throws DOMException;
925
926     /**
927     * See the list-style-image property definition in CSS2.
928     */
929     public String getListStyleImage();
930
931     /**
932     * See the list-style-image property definition in CSS2.
933     * @exception DOMException
934     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
935     *     unparsable.
936     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
937     */
938     public void setListStyleImage(String listStyleImage)
939         throws DOMException;
940
941     /**
942     * See the list-style-position property definition in CSS2.
943     */
944     public String getListStylePosition();
945
946     /**
947     * See the list-style-position property definition in CSS2.
948     * @exception DOMException
949     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
950     *     unparsable.
```

```
947     * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
948     */
949     public void setListStylePosition(String listStylePosition)
950         throws DOMException;
951
952     /**
953     * See the list-style-type property definition in CSS2.
954     */
955     public String getListStyleType();
956     /**
957     * See the list-style-type property definition in CSS2.
958     * @exception DOMException
959     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
960     *     unparsable.
961     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
962     */
963     public void setListStyleType(String listStyleType)
964         throws DOMException;
965
966     /**
967     * See the margin property definition in CSS2.
968     */
969     public String getMargin();
970     /**
971     * See the margin property definition in CSS2.
972     * @exception DOMException
973     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
974     *     unparsable.
975     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
976     */
977     public void setMargin(String margin)
978         throws DOMException;
979
980     /**
981     * See the margin-top property definition in CSS2.
982     */
983     public String getMarginTop();
984     /**
985     * See the margin-top property definition in CSS2.
986     * @exception DOMException
987     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
988     *     unparsable.
989     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
990     */
991     public void setMarginTop(String marginTop)
992         throws DOMException;
993
994     /**
995     * See the margin-right property definition in CSS2.
996     */
997     public String getMarginRight();
998     /**
999     * See the margin-right property definition in CSS2.
```

```
1000     * @exception DOMException
1001     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1002     *     unparsable.
1003     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1004     */
1005     public void setMarginRight(String marginRight)
1006         throws DOMException;
1007
1008     /**
1009     * See the margin-bottom property definition in CSS2.
1010     */
1011     public String getMarginBottom();
1012     /**
1013     * See the margin-bottom property definition in CSS2.
1014     * @exception DOMException
1015     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1016     *     unparsable.
1017     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1018     */
1019     public void setMarginBottom(String marginBottom)
1020         throws DOMException;
1021
1022     /**
1023     * See the margin-left property definition in CSS2.
1024     */
1025     public String getMarginLeft();
1026     /**
1027     * See the margin-left property definition in CSS2.
1028     * @exception DOMException
1029     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1030     *     unparsable.
1031     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1032     */
1033     public void setMarginLeft(String marginLeft)
1034         throws DOMException;
1035
1036     /**
1037     * See the marker-offset property definition in CSS2.
1038     */
1039     public String getMarkerOffset();
1040     /**
1041     * See the marker-offset property definition in CSS2.
1042     * @exception DOMException
1043     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1044     *     unparsable.
1045     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1046     */
1047     public void setMarkerOffset(String markerOffset)
1048         throws DOMException;
1049
1050     /**
1051     * See the marks property definition in CSS2.
1052     */
```



```
1106 /**
1107  * See the min-width property definition in CSS2.
1108  */
1109 public String getMinWidth();
1110 /**
1111  * See the min-width property definition in CSS2.
1112  * @exception DOMException
1113  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1114  *   unparsable.
1115  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1116  */
1117 public void setMinWidth(String minWidth)
1118     throws DOMException;
1119
1120 /**
1121  * See the orphans property definition in CSS2.
1122  */
1123 public String getOrphans();
1124 /**
1125  * See the orphans property definition in CSS2.
1126  * @exception DOMException
1127  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1128  *   unparsable.
1129  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1130  */
1131 public void setOrphans(String orphans)
1132     throws DOMException;
1133
1134 /**
1135  * See the outline property definition in CSS2.
1136  */
1137 public String getOutline();
1138 /**
1139  * See the outline property definition in CSS2.
1140  * @exception DOMException
1141  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1142  *   unparsable.
1143  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1144  */
1145 public void setOutline(String outline)
1146     throws DOMException;
1147
1148 /**
1149  * See the outline-color property definition in CSS2.
1150  */
1151 public String getOutlineColor();
1152 /**
1153  * See the outline-color property definition in CSS2.
1154  * @exception DOMException
1155  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1156  *   unparsable.
1157  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1158  */
```

```
1159     public void setOutlineColor(String outlineColor)
1160         throws DOMException;
1161
1162     /**
1163      * See the outline-style property definition in CSS2.
1164      */
1165     public String getOutlineStyle();
1166
1167     /**
1168      * See the outline-style property definition in CSS2.
1169      * @exception DOMException
1170      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1171      *     unparsable.
1172      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1173      */
1174     public void setOutlineStyle(String outlineStyle)
1175         throws DOMException;
1176
1177     /**
1178      * See the outline-width property definition in CSS2.
1179      */
1180     public String getOutlineWidth();
1181
1182     /**
1183      * See the outline-width property definition in CSS2.
1184      * @exception DOMException
1185      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1186      *     unparsable.
1187      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1188      */
1189     public void setOutlineWidth(String outlineWidth)
1190         throws DOMException;
1191
1192     /**
1193      * See the overflow property definition in CSS2.
1194      */
1195     public String getOverflow();
1196
1197     /**
1198      * See the overflow property definition in CSS2.
1199      * @exception DOMException
1200      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1201      *     unparsable.
1202      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1203      */
1204     public void setOverflow(String overflow)
1205         throws DOMException;
1206
1207     /**
1208      * See the padding property definition in CSS2.
1209      */
1210     public String getPadding();
1211
1212     /**
1213      * See the padding property definition in CSS2.
1214      * @exception DOMException
1215      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
```

```
1212     * unparsable.
1213     * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1214     */
1215     public void setPadding(String padding)
1216                             throws DOMException;
1217
1218     /**
1219     * See the padding-top property definition in CSS2.
1220     */
1221     public String getPaddingTop();
1222     /**
1223     * See the padding-top property definition in CSS2.
1224     * @exception DOMException
1225     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1226     *     unparsable.
1227     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1228     */
1229     public void setPaddingTop(String paddingTop)
1230                             throws DOMException;
1231
1232     /**
1233     * See the padding-right property definition in CSS2.
1234     */
1235     public String getPaddingRight();
1236     /**
1237     * See the padding-right property definition in CSS2.
1238     * @exception DOMException
1239     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1240     *     unparsable.
1241     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1242     */
1243     public void setPaddingRight(String paddingRight)
1244                             throws DOMException;
1245
1246     /**
1247     * See the padding-bottom property definition in CSS2.
1248     */
1249     public String getPaddingBottom();
1250     /**
1251     * See the padding-bottom property definition in CSS2.
1252     * @exception DOMException
1253     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1254     *     unparsable.
1255     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1256     */
1257     public void setPaddingBottom(String paddingBottom)
1258                             throws DOMException;
1259
1260     /**
1261     * See the padding-left property definition in CSS2.
1262     */
1263     public String getPaddingLeft();
1264     /**
```

```
1265     * See the padding-left property definition in CSS2.
1266     * @exception DOMException
1267     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1268     *     unparsable.
1269     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1270     */
1271     public void setPaddingLeft(String paddingLeft)
1272         throws DOMException;
1273
1274     /**
1275     * See the page property definition in CSS2.
1276     */
1277     public String getPage();
1278     /**
1279     * See the page property definition in CSS2.
1280     * @exception DOMException
1281     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1282     *     unparsable.
1283     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1284     */
1285     public void setPage(String page)
1286         throws DOMException;
1287
1288     /**
1289     * See the page-break-after property definition in CSS2.
1290     */
1291     public String getPageBreakAfter();
1292     /**
1293     * See the page-break-after property definition in CSS2.
1294     * @exception DOMException
1295     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1296     *     unparsable.
1297     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1298     */
1299     public void setPageBreakAfter(String pageBreakAfter)
1300         throws DOMException;
1301
1302     /**
1303     * See the page-break-before property definition in CSS2.
1304     */
1305     public String getPageBreakBefore();
1306     /**
1307     * See the page-break-before property definition in CSS2.
1308     * @exception DOMException
1309     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1310     *     unparsable.
1311     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1312     */
1313     public void setPageBreakBefore(String pageBreakBefore)
1314         throws DOMException;
1315
1316     /**
1317     * See the page-break-inside property definition in CSS2.
```



```
1318     */
1319     public String getPageBreakInside();
1320     /**
1321      * See the page-break-inside property definition in CSS2.
1322      * @exception DOMException
1323      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1324      *     unparsable.
1325      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1326      */
1327     public void setPageBreakInside(String pageBreakInside)
1328         throws DOMException;
1329
1330     /**
1331      * See the pause property definition in CSS2.
1332      */
1333     public String getPause();
1334     /**
1335      * See the pause property definition in CSS2.
1336      * @exception DOMException
1337      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1338      *     unparsable.
1339      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1340      */
1341     public void setPause(String pause)
1342         throws DOMException;
1343
1344     /**
1345      * See the pause-after property definition in CSS2.
1346      */
1347     public String getPauseAfter();
1348     /**
1349      * See the pause-after property definition in CSS2.
1350      * @exception DOMException
1351      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1352      *     unparsable.
1353      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1354      */
1355     public void setPauseAfter(String pauseAfter)
1356         throws DOMException;
1357
1358     /**
1359      * See the pause-before property definition in CSS2.
1360      */
1361     public String getPauseBefore();
1362     /**
1363      * See the pause-before property definition in CSS2.
1364      * @exception DOMException
1365      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1366      *     unparsable.
1367      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1368      */
1369     public void setPauseBefore(String pauseBefore)
1370         throws DOMException;
```

```
1371
1372 /**
1373  * See the pitch property definition in CSS2.
1374  */
1375 public String getPitch();
1376 /**
1377  * See the pitch property definition in CSS2.
1378  * @exception DOMException
1379  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1380  *   unparsable.
1381  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1382  */
1383 public void setPitch(String pitch)
1384                               throws DOMException;
1385
1386 /**
1387  * See the pitch-range property definition in CSS2.
1388  */
1389 public String getPitchRange();
1390 /**
1391  * See the pitch-range property definition in CSS2.
1392  * @exception DOMException
1393  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1394  *   unparsable.
1395  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1396  */
1397 public void setPitchRange(String pitchRange)
1398                               throws DOMException;
1399
1400 /**
1401  * See the play-during property definition in CSS2.
1402  */
1403 public String getPlayDuring();
1404 /**
1405  * See the play-during property definition in CSS2.
1406  * @exception DOMException
1407  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1408  *   unparsable.
1409  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1410  */
1411 public void setPlayDuring(String playDuring)
1412                               throws DOMException;
1413
1414 /**
1415  * See the position property definition in CSS2.
1416  */
1417 public String getPosition();
1418 /**
1419  * See the position property definition in CSS2.
1420  * @exception DOMException
1421  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1422  *   unparsable.
1423  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
```

```
1424     */
1425     public void setPosition(String position)
1426         throws DOMException;
1427
1428     /**
1429     * See the quotes property definition in CSS2.
1430     */
1431     public String getQuotes();
1432     /**
1433     * See the quotes property definition in CSS2.
1434     * @exception DOMException
1435     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1436     *     unparsable.
1437     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1438     */
1439     public void setQuotes(String quotes)
1440         throws DOMException;
1441
1442     /**
1443     * See the richness property definition in CSS2.
1444     */
1445     public String getRichness();
1446     /**
1447     * See the richness property definition in CSS2.
1448     * @exception DOMException
1449     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1450     *     unparsable.
1451     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1452     */
1453     public void setRichness(String richness)
1454         throws DOMException;
1455
1456     /**
1457     * See the right property definition in CSS2.
1458     */
1459     public String getRight();
1460     /**
1461     * See the right property definition in CSS2.
1462     * @exception DOMException
1463     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1464     *     unparsable.
1465     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1466     */
1467     public void setRight(String right)
1468         throws DOMException;
1469
1470     /**
1471     * See the size property definition in CSS2.
1472     */
1473     public String getSize();
1474     /**
1475     * See the size property definition in CSS2.
1476     * @exception DOMException
```

```
1477 * SYNTAX_ERR: Raised if the new value has a syntax error and is
1478 * unparsable.
1479 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1480 */
1481 public void setSize(String size)
1482                                     throws DOMException;
1483
1484 /**
1485 * See the speak property definition in CSS2.
1486 */
1487 public String getSpeak();
1488 /**
1489 * See the speak property definition in CSS2.
1490 * @exception DOMException
1491 * SYNTAX_ERR: Raised if the new value has a syntax error and is
1492 * unparsable.
1493 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1494 */
1495 public void setSpeak(String speak)
1496                                     throws DOMException;
1497
1498 /**
1499 * See the speak-header property definition in CSS2.
1500 */
1501 public String getSpeakHeader();
1502 /**
1503 * See the speak-header property definition in CSS2.
1504 * @exception DOMException
1505 * SYNTAX_ERR: Raised if the new value has a syntax error and is
1506 * unparsable.
1507 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1508 */
1509 public void setSpeakHeader(String speakHeader)
1510                                     throws DOMException;
1511
1512 /**
1513 * See the speak-numeral property definition in CSS2.
1514 */
1515 public String getSpeakNumeral();
1516 /**
1517 * See the speak-numeral property definition in CSS2.
1518 * @exception DOMException
1519 * SYNTAX_ERR: Raised if the new value has a syntax error and is
1520 * unparsable.
1521 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1522 */
1523 public void setSpeakNumeral(String speakNumeral)
1524                                     throws DOMException;
1525
1526 /**
1527 * See the speak-punctuation property definition in CSS2.
1528 */
1529 public String getSpeakPunctuation();
```

```
1530 /**
1531  * See the speak-punctuation property definition in CSS2.
1532  * @exception DOMException
1533  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1534  *   unparsable.
1535  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1536  */
1537 public void setSpeakPunctuation(String speakPunctuation)
1538     throws DOMException;
1539
1540 /**
1541  * See the speech-rate property definition in CSS2.
1542  */
1543 public String getSpeechRate();
1544 /**
1545  * See the speech-rate property definition in CSS2.
1546  * @exception DOMException
1547  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1548  *   unparsable.
1549  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1550  */
1551 public void setSpeechRate(String speechRate)
1552     throws DOMException;
1553
1554 /**
1555  * See the stress property definition in CSS2.
1556  */
1557 public String getStress();
1558 /**
1559  * See the stress property definition in CSS2.
1560  * @exception DOMException
1561  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1562  *   unparsable.
1563  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1564  */
1565 public void setStress(String stress)
1566     throws DOMException;
1567
1568 /**
1569  * See the table-layout property definition in CSS2.
1570  */
1571 public String getTableLayout();
1572 /**
1573  * See the table-layout property definition in CSS2.
1574  * @exception DOMException
1575  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1576  *   unparsable.
1577  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1578  */
1579 public void setTableLayout(String tableLayout)
1580     throws DOMException;
1581
1582 /**
```

```
1583     * See the text-align property definition in CSS2.
1584     */
1585     public String getTextAlign();
1586     /**
1587     * See the text-align property definition in CSS2.
1588     * @exception DOMException
1589     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1590     *     unparsable.
1591     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1592     */
1593     public void setTextAlign(String textAlign)
1594         throws DOMException;
1595
1596     /**
1597     * See the text-decoration property definition in CSS2.
1598     */
1599     public String getTextDecoration();
1600     /**
1601     * See the text-decoration property definition in CSS2.
1602     * @exception DOMException
1603     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1604     *     unparsable.
1605     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1606     */
1607     public void setTextDecoration(String textDecoration)
1608         throws DOMException;
1609
1610     /**
1611     * See the text-indent property definition in CSS2.
1612     */
1613     public String getTextIndent();
1614     /**
1615     * See the text-indent property definition in CSS2.
1616     * @exception DOMException
1617     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1618     *     unparsable.
1619     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1620     */
1621     public void setTextIndent(String textIndent)
1622         throws DOMException;
1623
1624     /**
1625     * See the text-shadow property definition in CSS2.
1626     */
1627     public String getTextShadow();
1628     /**
1629     * See the text-shadow property definition in CSS2.
1630     * @exception DOMException
1631     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1632     *     unparsable.
1633     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1634     */
1635     public void setTextShadow(String textShadow)
```

```
1636                                     throws DOMException;
1637
1638 /**
1639  * See the text-transform property definition in CSS2.
1640  */
1641 public String getTextTransform();
1642 /**
1643  * See the text-transform property definition in CSS2.
1644  * @exception DOMException
1645  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1646  *   unparsable.
1647  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1648  */
1649 public void setTextTransform(String textTransform)
1650                                     throws DOMException;
1651
1652 /**
1653  * See the top property definition in CSS2.
1654  */
1655 public String getTop();
1656 /**
1657  * See the top property definition in CSS2.
1658  * @exception DOMException
1659  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1660  *   unparsable.
1661  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1662  */
1663 public void setTop(String top)
1664                                     throws DOMException;
1665
1666 /**
1667  * See the unicode-bidi property definition in CSS2.
1668  */
1669 public String getUnicodeBidi();
1670 /**
1671  * See the unicode-bidi property definition in CSS2.
1672  * @exception DOMException
1673  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1674  *   unparsable.
1675  *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1676  */
1677 public void setUnicodeBidi(String unicodeBidi)
1678                                     throws DOMException;
1679
1680 /**
1681  * See the vertical-align property definition in CSS2.
1682  */
1683 public String getVerticalAlign();
1684 /**
1685  * See the vertical-align property definition in CSS2.
1686  * @exception DOMException
1687  *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1688  *   unparsable.
```

```
1689     * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1690     */
1691     public void setVerticalAlign(String verticalAlign)
1692         throws DOMException;
1693
1694     /**
1695     * See the visibility property definition in CSS2.
1696     */
1697     public String getVisibility();
1698     /**
1699     * See the visibility property definition in CSS2.
1700     * @exception DOMException
1701     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1702     *     unparsable.
1703     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1704     */
1705     public void setVisibility(String visibility)
1706         throws DOMException;
1707
1708     /**
1709     * See the voice-family property definition in CSS2.
1710     */
1711     public String getVoiceFamily();
1712     /**
1713     * See the voice-family property definition in CSS2.
1714     * @exception DOMException
1715     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1716     *     unparsable.
1717     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1718     */
1719     public void setVoiceFamily(String voiceFamily)
1720         throws DOMException;
1721
1722     /**
1723     * See the volume property definition in CSS2.
1724     */
1725     public String getVolume();
1726     /**
1727     * See the volume property definition in CSS2.
1728     * @exception DOMException
1729     *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1730     *     unparsable.
1731     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1732     */
1733     public void setVolume(String volume)
1734         throws DOMException;
1735
1736     /**
1737     * See the white-space property definition in CSS2.
1738     */
1739     public String getWhiteSpace();
1740     /**
1741     * See the white-space property definition in CSS2.
```



```
1742     * @exception DOMException
1743     *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1744     *   unparsable.
1745     *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1746     */
1747     public void setWhiteSpace(String whiteSpace)
1748         throws DOMException;
1749
1750     /**
1751     *   See the widows property definition in CSS2.
1752     */
1753     public String getWidows();
1754     /**
1755     *   See the widows property definition in CSS2.
1756     * @exception DOMException
1757     *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1758     *   unparsable.
1759     *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1760     */
1761     public void setWidows(String widows)
1762         throws DOMException;
1763
1764     /**
1765     *   See the width property definition in CSS2.
1766     */
1767     public String getWidth();
1768     /**
1769     *   See the width property definition in CSS2.
1770     * @exception DOMException
1771     *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1772     *   unparsable.
1773     *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1774     */
1775     public void setWidth(String width)
1776         throws DOMException;
1777
1778     /**
1779     *   See the word-spacing property definition in CSS2.
1780     */
1781     public String getWordSpacing();
1782     /**
1783     *   See the word-spacing property definition in CSS2.
1784     * @exception DOMException
1785     *   SYNTAX_ERR: Raised if the new value has a syntax error and is
1786     *   unparsable.
1787     *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1788     */
1789     public void setWordSpacing(String wordSpacing)
1790         throws DOMException;
1791
1792     /**
1793     *   See the z-index property definition in CSS2.
1794     */
```

```

1795     public String getZIndex();
1796     /**
1797      * See the z-index property definition in CSS2.
1798      * @exception DOMException
1799      *     SYNTAX_ERR: Raised if the new value has a syntax error and is
1800      *     unparsable.
1801      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
1802      */
1803     public void setZIndex(String zIndex)
1804                             throws DOMException;
1805
1806 }

```

## 38.2 CSSCharsetRule.java

Secuencia 362: org/w3c/dom/css/CSSCharsetRule.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR

```

```

38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  import org.w3c.dom.DOMException;
45
46  /**
47   * The CSSCharsetRule interface represents a @charset rule in a
48   * CSS style sheet. The value of the encoding attribute does
49   * not affect the encoding of text data in the DOM objects; this encoding is
50   * always UTF-16. After a stylesheet is loaded, the value of the
51   * encoding attribute is the value found in the
52   * @charset rule. If there was no @charset in the
53   * original document, then no CSSCharsetRule is created. The
54   * value of the encoding attribute may also be used as a hint
55   * for the encoding used on serialization of the style sheet.
56   * 

The value of the @charset rule (and therefore of the
57   * CSSCharsetRule) may not correspond to the encoding the
58   * document actually came in; character encoding information e.g. in an HTTP
59   * header, has priority (see CSS document representation) but this is not
60   * reflected in the CSSCharsetRule.


61   * 

See also the Document
62   * Object Model (DOM) Level 2 Style Specification.


63   * @since DOM Level 2
64   */
65  public interface CSSCharsetRule extends CSSRule {
66      /**
67       * The encoding information used in this @charset rule.
68       */
69      public String getEncoding();
70      /**
71       * The encoding information used in this @charset rule.
72       * @exception DOMException
73       *     SYNTAX_ERR: Raised if the specified encoding value has a syntax error
74       *     and is unparsable.
75       *     NO_MODIFICATION_ALLOWED_ERR: Raised if this encoding rule is
76       *     readonly.
77       */
78      public void setEncoding(String encoding)
79          throws DOMException;
80  }

```

### 38.3 CSSFontFaceRule.java

Secuencia 363: org/w3c/dom/css/CSSFontFaceRule.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *

```

```
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 /**
45  * The CSSFontFaceRule interface represents a @font-face rule in
46  * a CSS style sheet. The @font-face rule is used to hold a set
47  * of font descriptions.
48  * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
49  *   Object Model (DOM) Level 2 Style Specification</a>.
50  * @since DOM Level 2
51  */
52 public interface CSSFontFaceRule extends CSSRule {
53     /**
54      * The declaration-block of this rule.
55      */
56     public CSSStyleDeclaration getStyle();
57 }
```

**38.4** CSSImportRule.java

Secuencia 364: org/w3c/dom/css/CSSImportRule.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.stylesheets.MediaList;
45
46 /**
47 * The CSSImportRule interface represents a @import rule within
48 * a CSS style sheet. The @import rule is used to import style
49 * rules from other style sheets.
```

```

50 * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
    Object Model (DOM) Level 2 Style Specification</a>.
51 * @since DOM Level 2
52 */
53 public interface CSSImportRule extends CSSRule {
54     /**
55      * The location of the style sheet to be imported. The attribute will not
56      * contain the <code>"url(...)"</code> specifier around the URI.
57      */
58     public String getHref();
59
60     /**
61      * A list of media types for which this style sheet may be used.
62      */
63     public MediaList getMedia();
64
65     /**
66      * The style sheet referred to by this rule, if it has been loaded. The
67      * value of this attribute is <code>null</code> if the style sheet has
68      * not yet been loaded or if it will not be loaded (e.g. if the style
69      * sheet is for a media type not supported by the user agent).
70      */
71     public CSSStyleSheet getStyleSheet();
72
73 }

```

### 38.5 CSSMediaRule.java

Secuencia 365: org/w3c/dom/css/CSSMediaRule.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24

```

```
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  import org.w3c.dom.DOMException;
45  import org.w3c.dom.stylesheets.MediaList;
46
47  /**
48   * The CSSMediaRule interface represents a @media rule in a CSS
49   * style sheet. A @media rule can be used to delimit style
50   * rules for specific media types.
51   * 

See also the Document
52   * Object Model (DOM) Level 2 Style Specification.


53   * @since DOM Level 2
54   */
55  public interface CSSMediaRule extends CSSRule {
56      /**
57       * A list of media types for this rule.
58       */
59      public MediaList getMedia();
60
61      /**
62       * A list of all CSS rules contained within the media block.
63       */
64      public CSSRuleList getCssRules();
65
66      /**
67       * Used to insert a new rule into the media block.
68       * @param rule The parsable text representing the rule. For rule sets
69       * this contains both the selector and the style declaration. For
70       * at-rules, this specifies both the at-identifier and the rule
71       * content.
72       * @param index The index within the media block's rule collection of
73       * the rule before which to insert the specified rule. If the
74       * specified index is equal to the length of the media blocks's rule
75       * collection, the rule will be added to the end of the media block.
76       * @return The index within the media block's rule collection of the
77       * newly inserted rule.
```

```

77      * @exception DOMException
78      *   HIERARCHY_REQUEST_ERR: Raised if the rule cannot be inserted at the
79      *   specified index, e.g., if an <code>@import</code> rule is inserted
80      *   after a standard rule set or other at-rule.
81      *   <br>INDEX_SIZE_ERR: Raised if the specified index is not a valid
82      *   insertion point.
83      *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this media rule is
84      *   readonly.
85      *   <br>SYNTAX_ERR: Raised if the specified rule has a syntax error and
86      *   is unparsable.
87      */
88      public int insertRule(String rule,
89                          int index)
90                          throws DOMException;
91
92      /**
93      * Used to delete a rule from the media block.
94      * @param index The index within the media block's rule collection of
95      * the rule to remove.
96      * @exception DOMException
97      *   INDEX_SIZE_ERR: Raised if the specified index does not correspond to
98      *   a rule in the media rule list.
99      *   <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this media rule is
100     *   readonly.
101     */
102     public void deleteRule(int index)
103                         throws DOMException;
104
105 }

```

### 38.6 CSSPageRule.java

Secuencia 366: [org.w3c/dom/css/CSSPageRule.java](http://org.w3c/dom/css/CSSPageRule.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```



```

21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  import org.w3c.dom.DOMException;
45
46  /**
47   * The CSSPageRule interface represents a @page rule within a
48   * CSS style sheet. The @page rule is used to specify the
49   * dimensions, orientation, margins, etc. of a page box for paged media.
50   * 

See also the Document
51   * Object Model (DOM) Level 2 Style Specification.


52   * @since DOM Level 2
53   */
54  public interface CSSPageRule extends CSSRule {
55      /**
56       * The parsable textual representation of the page selector for the rule.
57       */
58      public String getSelectorText();
59      /**
60       * The parsable textual representation of the page selector for the rule.
61       * @exception DOMException
62       *     SYNTAX_ERR: Raised if the specified CSS string value has a syntax
63       *     error and is unparseable.
64       *     NO_MODIFICATION_ALLOWED_ERR: Raised if this rule is readonly.
65       */
66      public void setSelectorText(String selectorText)
67          throws DOMException;
68      /**
69       * The declaration-block of this rule.
70       */
71      public CSSStyleDeclaration getStyle();
72

```

73 }

### 38.7 CSSPrimitiveValue.java

Secuencia 367: org/w3c/dom/css/CSSPrimitiveValue.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.DOMException;
45
46 /**
47 * The CSSPrimitiveValue interface represents a single CSS value
48 * . This interface may be used to determine the value of a specific style
```

```

49  * property currently set in a block or to set a specific style property
50  * explicitly within the block. An instance of this interface might be
51  * obtained from the <code>getPropertyCSSValue</code> method of the
52  * <code>CSSStyleDeclaration</code> interface. A
53  * <code>CSSPrimitiveValue</code> object only occurs in a context of a CSS
54  * property.
55  * <p> Conversions are allowed between absolute values (from millimeters to
56  * centimeters, from degrees to radians, and so on) but not between relative
57  * values. (For example, a pixel value cannot be converted to a centimeter
58  * value.) Percentage values can't be converted since they are relative to
59  * the parent value (or another property value). There is one exception for
60  * color percentage values: since a color percentage value is relative to
61  * the range 0-255, a color percentage value can be converted to a number;
62  * (see also the <code>RGBColor</code> interface).
63  * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
    Object Model (DOM) Level 2 Style Specification</a>.
64  * @since DOM Level 2
65  */
66  public interface CSSPrimitiveValue extends CSSValue {
67      // UnitTypes
68      /**
69       * The value is not a recognized CSS2 value. The value can only be
70       * obtained by using the <code>cssText</code> attribute.
71       */
72      public static final short CSS_UNKNOWN          = 0;
73      /**
74       * The value is a simple number. The value can be obtained by using the
75       * <code>getFloatValue</code> method.
76       */
77      public static final short CSS_NUMBER           = 1;
78      /**
79       * The value is a percentage. The value can be obtained by using the
80       * <code>getFloatValue</code> method.
81       */
82      public static final short CSS_PERCENTAGE       = 2;
83      /**
84       * The value is a length (ems). The value can be obtained by using the
85       * <code>getFloatValue</code> method.
86       */
87      public static final short CSS_EMS              = 3;
88      /**
89       * The value is a length (exs). The value can be obtained by using the
90       * <code>getFloatValue</code> method.
91       */
92      public static final short CSS_EXS              = 4;
93      /**
94       * The value is a length (px). The value can be obtained by using the
95       * <code>getFloatValue</code> method.
96       */
97      public static final short CSS_PX               = 5;
98      /**
99       * The value is a length (cm). The value can be obtained by using the
100     * <code>getFloatValue</code> method.

```

```
101     */
102     public static final short CSS_CM                      = 6;
103     /**
104      * The value is a length (mm). The value can be obtained by using the
105      * <code>getFloatValue</code> method.
106      */
107     public static final short CSS_MM                      = 7;
108     /**
109      * The value is a length (in). The value can be obtained by using the
110      * <code>getFloatValue</code> method.
111      */
112     public static final short CSS_IN                      = 8;
113     /**
114      * The value is a length (pt). The value can be obtained by using the
115      * <code>getFloatValue</code> method.
116      */
117     public static final short CSS_PT                      = 9;
118     /**
119      * The value is a length (pc). The value can be obtained by using the
120      * <code>getFloatValue</code> method.
121      */
122     public static final short CSS_PC                      = 10;
123     /**
124      * The value is an angle (deg). The value can be obtained by using the
125      * <code>getFloatValue</code> method.
126      */
127     public static final short CSS_DEG                    = 11;
128     /**
129      * The value is an angle (rad). The value can be obtained by using the
130      * <code>getFloatValue</code> method.
131      */
132     public static final short CSS_RAD                    = 12;
133     /**
134      * The value is an angle (grad). The value can be obtained by using the
135      * <code>getFloatValue</code> method.
136      */
137     public static final short CSS_GRAD                   = 13;
138     /**
139      * The value is a time (ms). The value can be obtained by using the
140      * <code>getFloatValue</code> method.
141      */
142     public static final short CSS_MS                     = 14;
143     /**
144      * The value is a time (s). The value can be obtained by using the
145      * <code>getFloatValue</code> method.
146      */
147     public static final short CSS_S                     = 15;
148     /**
149      * The value is a frequency (Hz). The value can be obtained by using the
150      * <code>getFloatValue</code> method.
151      */
152     public static final short CSS_HZ                     = 16;
153     /**
```

```
154     * The value is a frequency (kHz). The value can be obtained by using the
155     * <code>getFloatValue</code> method.
156     */
157     public static final short CSS_KHZ = 17;
158     /**
159     * The value is a number with an unknown dimension. The value can be
160     * obtained by using the <code>getFloatValue</code> method.
161     */
162     public static final short CSS_DIMENSION = 18;
163     /**
164     * The value is a STRING. The value can be obtained by using the
165     * <code>getStringValue</code> method.
166     */
167     public static final short CSS_STRING = 19;
168     /**
169     * The value is a URI. The value can be obtained by using the
170     * <code>getStringValue</code> method.
171     */
172     public static final short CSS_URI = 20;
173     /**
174     * The value is an identifier. The value can be obtained by using the
175     * <code>getStringValue</code> method.
176     */
177     public static final short CSS_IDENT = 21;
178     /**
179     * The value is a attribute function. The value can be obtained by using
180     * the <code>getStringValue</code> method.
181     */
182     public static final short CSS_ATTR = 22;
183     /**
184     * The value is a counter or counters function. The value can be obtained
185     * by using the <code>getCounterValue</code> method.
186     */
187     public static final short CSS_COUNTER = 23;
188     /**
189     * The value is a rect function. The value can be obtained by using the
190     * <code>getRectValue</code> method.
191     */
192     public static final short CSS_RECT = 24;
193     /**
194     * The value is a RGB color. The value can be obtained by using the
195     * <code>getRGBColorValue</code> method.
196     */
197     public static final short CSS_RGBCOLOR = 25;
198
199     /**
200     * The type of the value as defined by the constants specified above.
201     */
202     public short getPrimitiveType();
203
204     /**
205     * A method to set the float value with a specified unit. If the property
206     * attached with this value can not accept the specified unit or the
```

```

207 * float value, the value will be unchanged and a
208 * <code>DOMException</code> will be raised.
209 * @param unitType A unit code as defined above. The unit code can only
210 * be a float unit type (i.e. <code>CSS_NUMBER</code>,
211 * <code>CSS_PERCENTAGE</code>, <code>CSS_EMS</code>,
212 * <code>CSS_EXS</code>, <code>CSS_PX</code>, <code>CSS_CM</code>,
213 * <code>CSS_MM</code>, <code>CSS_IN</code>, <code>CSS_PT</code>,
214 * <code>CSS_PC</code>, <code>CSS_DEG</code>, <code>CSS_RAD</code>,
215 * <code>CSS_GRAD</code>, <code>CSS_MS</code>, <code>CSS_S</code>,
216 * <code>CSS_HZ</code>, <code>CSS_KHZ</code>,
217 * <code>CSS_DIMENSION</code>).
218 * @param floatValue The new float value.
219 * @exception DOMException
220 * INVALID_ACCESS_ERR: Raised if the attached property doesn't support
221 * the float value or the unit type.
222 * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
223 */
224 public void setFloatValue(short unitType,
225                          float floatValue)
226     throws DOMException;
227
228 /**
229 * This method is used to get a float value in a specified unit. If this
230 * CSS value doesn't contain a float value or can't be converted into
231 * the specified unit, a <code>DOMException</code> is raised.
232 * @param unitType A unit code to get the float value. The unit code can
233 * only be a float unit type (i.e. <code>CSS_NUMBER</code>,
234 * <code>CSS_PERCENTAGE</code>, <code>CSS_EMS</code>,
235 * <code>CSS_EXS</code>, <code>CSS_PX</code>, <code>CSS_CM</code>,
236 * <code>CSS_MM</code>, <code>CSS_IN</code>, <code>CSS_PT</code>,
237 * <code>CSS_PC</code>, <code>CSS_DEG</code>, <code>CSS_RAD</code>,
238 * <code>CSS_GRAD</code>, <code>CSS_MS</code>, <code>CSS_S</code>,
239 * <code>CSS_HZ</code>, <code>CSS_KHZ</code>,
240 * <code>CSS_DIMENSION</code>).
241 * @return The float value in the specified unit.
242 * @exception DOMException
243 * INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a float
244 * value or if the float value can't be converted into the specified
245 * unit.
246 */
247 public float getFloatValue(short unitType)
248     throws DOMException;
249
250 /**
251 * A method to set the string value with the specified unit. If the
252 * property attached to this value can't accept the specified unit or
253 * the string value, the value will be unchanged and a
254 * <code>DOMException</code> will be raised.
255 * @param stringType A string code as defined above. The string code can
256 * only be a string unit type (i.e. <code>CSS_STRING</code>,
257 * <code>CSS_URI</code>, <code>CSS_IDENT</code>, and
258 * <code>CSS_ATTR</code>).
259 * @param stringValue The new string value.

```

```

260     * @exception DOMException
261     *     INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a string
262     *     value or if the string value can't be converted into the specified
263     *     unit.
264     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.
265     */
266     public void setStringValue(short stringType,
267                               String stringValue)
268         throws DOMException;
269
270     /**
271     * This method is used to get the string value. If the CSS value doesn't
272     * contain a string value, a <code>DOMException</code> is raised. Some
273     * properties (like 'font-family' or 'voice-family') convert a
274     * whitespace separated list of idents to a string.
275     * @return The string value in the current unit. The current
276     * <code>primitiveType</code> can only be a string unit type (i.e.
277     * <code>CSS_STRING</code>, <code>CSS_URI</code>,
278     * <code>CSS_IDENT</code> and <code>CSS_ATTR</code>).
279     * @exception DOMException
280     *     INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a string
281     *     value.
282     */
283     public String getStringValue()
284         throws DOMException;
285
286     /**
287     * This method is used to get the Counter value. If this CSS value
288     * doesn't contain a counter value, a <code>DOMException</code> is
289     * raised. Modification to the corresponding style property can be
290     * achieved using the <code>Counter</code> interface.
291     * @return The Counter value.
292     * @exception DOMException
293     *     INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a
294     *     Counter value (e.g. this is not <code>CSS_COUNTER</code>).
295     */
296     public Counter getCounterValue()
297         throws DOMException;
298
299     /**
300     * This method is used to get the Rect value. If this CSS value doesn't
301     * contain a rect value, a <code>DOMException</code> is raised.
302     * Modification to the corresponding style property can be achieved
303     * using the <code>Rect</code> interface.
304     * @return The Rect value.
305     * @exception DOMException
306     *     INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a Rect
307     *     value. (e.g. this is not <code>CSS_RECT</code>).
308     */
309     public Rect getRectValue()
310         throws DOMException;
311
312     /**

```

```

313     * This method is used to get the RGB color. If this CSS value doesn't
314     * contain a RGB color value, a <code>DOMException</code> is raised.
315     * Modification to the corresponding style property can be achieved
316     * using the <code>RGBColor</code> interface.
317     * @return the RGB color value.
318     * @exception DOMException
319     *     INVALID_ACCESS_ERR: Raised if the attached property can't return a
320     *     RGB color value (e.g. this is not <code>CSS_RGBCOLOR</code>).
321     */
322     public RGBColor getRGBColorValue()
323         throws DOMException;
324
325 }

```

### 38.8 CSSRule.java

Secuencia 368: [org/w3c/dom/css/CSSRule.java](http://org/w3c/dom/css/CSSRule.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even

```



```
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  import org.w3c.dom.DOMException;
45
46  /**
47   * The CSSRule interface is the abstract base interface for any
48   * type of CSS statement. This includes both rule sets and at-rules. An
49   * implementation is expected to preserve all rules specified in a CSS style
50   * sheet, even if the rule is not recognized by the parser. Unrecognized
51   * rules are represented using the CSSUnknownRule interface.
52   * 

See also the Document Object Model \(DOM\) Level 2 Style Specification.


53   * @since DOM Level 2
54   */
55  public interface CSSRule {
56      // RuleType
57      /**
58       * The rule is a CSSUnknownRule.
59       */
60      public static final short UNKNOWN_RULE = 0;
61      /**
62       * The rule is a CSSStyleRule.
63       */
64      public static final short STYLE_RULE = 1;
65      /**
66       * The rule is a CSSCharsetRule.
67       */
68      public static final short CHARSET_RULE = 2;
69      /**
70       * The rule is a CSSImportRule.
71       */
72      public static final short IMPORT_RULE = 3;
73      /**
74       * The rule is a CSSMediaRule.
75       */
76      public static final short MEDIA_RULE = 4;
77      /**
78       * The rule is a CSSFontFaceRule.
79       */
80      public static final short FONT_FACE_RULE = 5;
81      /**
82       * The rule is a CSSPageRule.
83       */
84      public static final short PAGE_RULE = 6;
85
86      /**
87       * The type of the rule, as defined above. The expectation is that
88       * binding-specific casting methods can be used to cast down from an
```

```

89     * instance of the <code>CSSRule</code> interface to the specific
90     * derived interface implied by the <code>type</code>.
91     */
92     public short getType();
93
94     /**
95     * The parsable textual representation of the rule. This reflects the
96     * current state of the rule and not its initial value.
97     */
98     public String getCssText();
99     /**
100    * The parsable textual representation of the rule. This reflects the
101    * current state of the rule and not its initial value.
102    * @exception DOMException
103    *     SYNTAX_ERR: Raised if the specified CSS string value has a syntax
104    *     error and is unparsable.
105    *     <br>INVALID_MODIFICATION_ERR: Raised if the specified CSS string
106    *     value represents a different type of rule than the current one.
107    *     <br>HIERARCHY_REQUEST_ERR: Raised if the rule cannot be inserted at
108    *     this point in the style sheet.
109    *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if the rule is readonly.
110    */
111     public void setCssText(String cssText)
112         throws DOMException;
113
114     /**
115     * The style sheet that contains this rule.
116     */
117     public CSSStyleSheet getParentStyleSheet();
118
119     /**
120     * If this rule is contained inside another rule (e.g. a style rule
121     * inside an @media block), this is the containing rule. If this rule is
122     * not nested inside any other rules, this returns <code>null</code>.
123     */
124     public CSSRule getParentRule();
125
126 }

```

### 38.9 CSSRuleList.java

Secuencia 369: [org.w3c/dom/css/CSSRuleList.java](http://org.w3c/dom/css/CSSRuleList.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *

```

```
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  /**
45   * The CSSRuleList interface provides the abstraction of an
46   * ordered collection of CSS rules.
47   * 

The items in the CSSRuleList are accessible via an
48   * integral index, starting from 0.
49   * 

See also the Document
49   * Object Model (DOM) Level 2 Style Specification.


50   * @since DOM Level 2
51   */
52  public interface CSSRuleList {
53      /**
54       * The number of CSSRules in the list. The range of valid
55       * child rule indices is 0 to length-1
56       * inclusive.
57       */
58      public int getLength();
59
60      /**
61       * Used to retrieve a CSS rule by ordinal index. The order in this
62       * collection represents the order of the rules in the CSS style sheet.
63       * If index is greater than or equal to the number of rules in the list,


```

```

64     * this returns <code>null</code>.
65     * @param index Index into the collection
66     * @return The style rule at the <code>index</code> position in the
67     *         <code>CSSRuleList</code>, or <code>null</code> if that is not a
68     *         valid index.
69     */
70     public CSSRule item(int index);
71
72 }

```

### 38.10 CSSStyleDeclaration.java

Secuencia 370: org/w3c/dom/css/CSSStyleDeclaration.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */

```

```

41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.DOMException;
45
46 /**
47  * The CSSStyleDeclaration interface represents a single CSS
48  * declaration block. This interface may be used to determine the style
49  * properties currently set in a block or to set style properties explicitly
50  * within the block.
51  * <p> While an implementation may not recognize all CSS properties within a
52  * CSS declaration block, it is expected to provide access to all specified
53  * properties in the style sheet through the CSSStyleDeclaration
54  * interface. Furthermore, implementations that support a specific level of
55  * CSS should correctly handle CSS shorthand properties for that level. For
56  * a further discussion of shorthand properties, see the
57  * CSS2Properties interface.
58  * <p> This interface is also used to provide a read-only access to the
59  * computed values of an element. See also the ViewCSS
60  * interface. The CSS Object Model doesn't provide an access to the
61  * specified or actual values of the CSS cascade.
62  * <p> See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
        Object Model (DOM) Level 2 Style Specification</a>.
63  * @since DOM Level 2
64  */
65 public interface CSSStyleDeclaration {
66     /**
67      * The parsable textual representation of the declaration block
68      * (excluding the surrounding curly braces). Setting this attribute will
69      * result in the parsing of the new value and resetting of all the
70      * properties in the declaration block including the removal or addition
71      * of properties.
72      */
73     public String getCssText();
74     /**
75      * The parsable textual representation of the declaration block
76      * (excluding the surrounding curly braces). Setting this attribute will
77      * result in the parsing of the new value and resetting of all the
78      * properties in the declaration block including the removal or addition
79      * of properties.
80      * @exception DOMException
81      *     SYNTAX_ERR: Raised if the specified CSS string value has a syntax
82      *     error and is unparsable.
83      *     NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is
84      *     readonly or a property is readonly.
85      */
86     public void setCssText(String cssText)
87         throws DOMException;
88
89     /**
90      * Used to retrieve the value of a CSS property if it has been explicitly
91      * set within this declaration block.
92      * @param propertyName The name of the CSS property. See the CSS

```

```

93     *   property index.
94     *   @return Returns the value of the property if it has been explicitly
95     *   set for this declaration block. Returns the empty string if the
96     *   property has not been set.
97     */
98     public String getPropertyValue(String propertyName);
99
100    /**
101     *   Used to retrieve the object representation of the value of a CSS
102     *   property if it has been explicitly set within this declaration block.
103     *   This method returns null if the property is a shorthand
104     *   property. Shorthand property values can only be accessed and modified
105     *   as strings, using the getPropertyValue and
106     *   setProperty methods.
107     *   @param propertyName The name of the CSS property. See the CSS
108     *   property index.
109     *   @return Returns the value of the property if it has been explicitly
110     *   set for this declaration block. Returns null if the
111     *   property has not been set.
112     */
113     public CSSValue getPropertyValueCSSValue(String propertyName);
114
115    /**
116     *   Used to remove a CSS property if it has been explicitly set within
117     *   this declaration block.
118     *   @param propertyName The name of the CSS property. See the CSS
119     *   property index.
120     *   @return Returns the value of the property if it has been explicitly
121     *   set for this declaration block. Returns the empty string if the
122     *   property has not been set or the property name does not correspond
123     *   to a known CSS property.
124     *   @exception DOMException
125     *   NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly
126     *   or the property is readonly.
127     */
128     public String removeProperty(String propertyName)
129         throws DOMException;
130
131    /**
132     *   Used to retrieve the priority of a CSS property (e.g. the
133     *   "important" qualifier) if the priority has been
134     *   explicitly set in this declaration block.
135     *   @param propertyName The name of the CSS property. See the CSS
136     *   property index.
137     *   @return A string representing the priority (e.g.
138     *   "important") if the property has been explicitly set
139     *   in this declaration block and has a priority specified. The empty
140     *   string otherwise.
141     */
142     public String getPropertyPriority(String propertyName);
143
144    /**
145     *   Used to set a property value and priority within this declaration

```

```

146 * block. <code>setProperty</code> permits to modify a property or add a
147 * new one in the declaration block. Any call to this method may modify
148 * the order of properties in the <code>item</code> method.
149 * @param propertyName The name of the CSS property. See the CSS
150 * property index.
151 * @param value The new value of the property.
152 * @param priority The new priority of the property (e.g.
153 * <code>"important"</code>) or the empty string if none.
154 * @exception DOMException
155 * SYNTAX_ERR: Raised if the specified value has a syntax error and is
156 * unparsable.
157 * NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is
158 * readonly or the property is readonly.
159 */
160 public void setProperty(String propertyName,
161                        String value,
162                        String priority)
163     throws DOMException;
164
165 /**
166 * The number of properties that have been explicitly set in this
167 * declaration block. The range of valid indices is 0 to length-1
168 * inclusive.
169 */
170 public int getLength();
171
172 /**
173 * Used to retrieve the properties that have been explicitly set in this
174 * declaration block. The order of the properties retrieved using this
175 * method does not have to be the order in which they were set. This
176 * method can be used to iterate over all properties in this declaration
177 * block.
178 * @param index Index of the property name to retrieve.
179 * @return The name of the property at this ordinal position. The empty
180 * string if no property exists at this position.
181 */
182 public String item(int index);
183
184 /**
185 * The CSS rule that contains this declaration block or <code>null</code>
186 * if this <code>CSSStyleDeclaration</code> is not attached to a
187 * <code>CSSRule</code>.
188 */
189 public CSSRule getParentRule();
190
191 }

```

### 38.11 CSSStyleRule.java

Secuencia 371: org/w3c/dom/css/CSSStyleRule.java

```

1 /*
2 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3 *

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.DOMException;
45
46 /**
47  * The CSSStyleRule interface represents a single rule set in a
48  * CSS style sheet.
49  * 

See also the Document
49  * Object Model (DOM) Level 2 Style Specification.


50  * @since DOM Level 2
51  */
52 public interface CSSStyleRule extends CSSRule {
53     /**
54      * The textual representation of the selector for the rule set. The
55      * implementation may have stripped out insignificant whitespace while
```



```

56     * parsing the selector.
57     */
58     public String getSelectorText();
59     /**
60     * The textual representation of the selector for the rule set. The
61     * implementation may have stripped out insignificant whitespace while
62     * parsing the selector.
63     * @exception DOMException
64     *     SYNTAX_ERR: Raised if the specified CSS string value has a syntax
65     *     error and is unparsable.
66     *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this rule is readonly.
67     */
68     public void setSelectorText(String selectorText)
69         throws DOMException;
70
71     /**
72     * The declaration-block of this rule set.
73     */
74     public CSSStyleDeclaration getStyle();
75
76 }

```

### 38.12 CSSStyleSheet.java

Secuencia 372: org/w3c/dom/css/CSSStyleSheet.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *

```

```

29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.DOMException;
45 import org.w3c.dom.stylesheets.StyleSheet;
46
47 /**
48  * The CSSStyleSheet interface is a concrete interface used to
49  * represent a CSS style sheet i.e., a style sheet whose content type is
50  * "text/css".
51  * 

See also the Document
52  * Object Model (DOM) Level 2 Style Specification

.
53  * @since DOM Level 2
54  */
55 public interface CSSStyleSheet extends StyleSheet {
56     /**
57      * If this style sheet comes from an @import rule, the
58      * ownerRule attribute will contain the
59      * CSSImportRule. In that case, the ownerNode
60      * attribute in the StyleSheet interface will be
61      * null. If the style sheet comes from an element or a
62      * processing instruction, the ownerRule attribute will be
63      * null and the ownerNode attribute will
64      * contain the Node.
65      */
66     public CSSRule getOwnerRule();
67
68     /**
69      * The list of all CSS rules contained within the style sheet. This
70      * includes both rule sets and at-rules.
71      */
72     public CSSRuleList getCssRules();
73
74     /**
75      * Used to insert a new rule into the style sheet. The new rule now
76      * becomes part of the cascade.
77      * @param rule The parsable text representing the rule. For rule sets
78      * this contains both the selector and the style declaration. For
79      * at-rules, this specifies both the at-identifier and the rule
80      * content.
81      * @param index The index within the style sheet's rule list of the rule

```

```

81      * before which to insert the specified rule. If the specified index
82      * is equal to the length of the style sheet's rule collection, the
83      * rule will be added to the end of the style sheet.
84      * @return The index within the style sheet's rule collection of the
85      * newly inserted rule.
86      * @exception DOMException
87      * HIERARCHY_REQUEST_ERR: Raised if the rule cannot be inserted at the
88      * specified index e.g. if an <code>@import</code> rule is inserted
89      * after a standard rule set or other at-rule.
90      * <br>INDEX_SIZE_ERR: Raised if the specified index is not a valid
91      * insertion point.
92      * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this style sheet is
93      * readonly.
94      * <br>SYNTAX_ERR: Raised if the specified rule has a syntax error and
95      * is unparsable.
96      */
97      public int insertRule(String rule,
98                          int index)
99                          throws DOMException;
100
101      /**
102      * Used to delete a rule from the style sheet.
103      * @param index The index within the style sheet's rule list of the rule
104      * to remove.
105      * @exception DOMException
106      * INDEX_SIZE_ERR: Raised if the specified index does not correspond to
107      * a rule in the style sheet's rule list.
108      * <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this style sheet is
109      * readonly.
110      */
111      public void deleteRule(int index)
112                          throws DOMException;
113
114  }

```

## 38.13 CSSUnknownRule.java

Secuencia 373: org/w3c/dom/css/CSSUnknownRule.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  /**
45   * The CSSUnknownRule interface represents an at-rule not
46   * supported by this user agent.
47   * 

See also the Document
48   * Object Model (DOM) Level 2 Style Specification.


49   * @since DOM Level 2
50   */
51  public interface CSSUnknownRule extends CSSRule {

```

### 38.14 CSSValue.java

Secuencia 374: org/w3c/dom/css/CSSValue.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  import org.w3c.dom.DOMException;
45
46  /**
47   * The CSSValue interface represents a simple or a complex
48   * value. A CSSValue object only occurs in a context of a CSS
49   * property.
50   * 

See also the Document
51   * Object Model (DOM) Level 2 Style Specification.


51   * @since DOM Level 2
52   */
53  public interface CSSValue {
54      // UnitTypes
55      /**
56       * The value is inherited and the cssText contains "inherit".
57       */
58      public static final short CSS_INHERIT = 0;
59      /**
60       * The value is a primitive value and an instance of the
61       * CSSPrimitiveValue interface can be obtained by using
62       * binding-specific casting methods on this instance of the
63       * CSSValue interface.
64       */
65  }
```

```

65     public static final short CSS_PRIMITIVE_VALUE      = 1;
66     /**
67      * The value is a CSSValue list and an instance of the
68      * CSSValueList interface can be obtained by using
69      * binding-specific casting methods on this instance of the
70      * CSSValue interface.
71      */
72     public static final short CSS_VALUE_LIST           = 2;
73     /**
74      * The value is a custom value.
75      */
76     public static final short CSS_CUSTOM               = 3;
77
78     /**
79      * A string representation of the current value.
80      */
81     public String getCssText();
82     /**
83      * A string representation of the current value.
84      * @exception DOMException
85      *     SYNTAX_ERR: Raised if the specified CSS string value has a syntax
86      *     error (according to the attached property) or is unparsable.
87      *     INVALID_MODIFICATION_ERR: Raised if the specified CSS string
88      *     value represents a different type of values than the values allowed
89      *     by the CSS property.
90      *     NO_MODIFICATION_ALLOWED_ERR: Raised if this value is readonly.
91      */
92     public void setCssText(String cssText)
93         throws DOMException;
94
95     /**
96      * A code defining the type of the value as defined above.
97      */
98     public short getCssValueType();
99
100 }

```

### 38.15 CSSValueList.java

Secuencia 375: [org.w3c/dom/css/CSSValueList.java](http://org.w3c/dom/css/CSSValueList.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```

14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  /**
45   * The CSSValueList interface provides the abstraction of an
46   * ordered collection of CSS values.
47   * 

Some properties allow an empty list into their syntax. In that case,
48   * these properties take the none identifier. So, an empty list
49   * means that the property has the value none.


50   * 

The items in the CSSValueList are accessible via an
51   * integral index, starting from 0.


52   * 

See also the http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113 Document
53   * Object Model (DOM) Level 2 Style Specification.


54   * @since DOM Level 2
55   */
56  public interface CSSValueList extends CSSValue {
57      /**
58       * The number of CSSValues in the list. The range of valid
59       * values of the indices is 0 to length-1
60       * inclusive.
61       */
62      public int getLength();
63
64      /**
65       * Used to retrieve a CSSValue by ordinal index. The order in
66       * this collection represents the order of the values in the CSS style

```

```
66      * property. If index is greater than or equal to the number of values
67      * in the list, this returns <code>null</code>.
68      * @param index Index into the collection.
69      * @return The <code>CSSValue</code> at the <code>index</code> position
70      * in the <code>CSSValueList</code>, or <code>null</code> if that is
71      * not a valid index.
72      */
73      public CSSValue item(int index);
74
75 }
```

### 38.16 Counter.java

Secuencia 376: org/w3c/dom/css/Counter.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
```



```
40  */
41
42  package org.w3c.dom.css;
43
44  /**
45   * The <code>Counter</code> interface is used to represent any counter or
46   * counters function value. This interface reflects the values in the
47   * underlying style property.
48   * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
      Object Model (DOM) Level 2 Style Specification</a>.
49   * @since DOM Level 2
50   */
51  public interface Counter {
52      /**
53       * This attribute is used for the identifier of the counter.
54       */
55      public String getIdentifier();
56
57      /**
58       * This attribute is used for the style of the list.
59       */
60      public String getListStyle();
61
62      /**
63       * This attribute is used for the separator of the nested counters.
64       */
65      public String getSeparator();
66  }
67 }
```

### 38.17 DOMImplementationCSS.java

Secuencia 377: org/w3c/dom/css/DOMImplementationCSS.java

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
```

```

21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  import org.w3c.dom.DOMImplementation;
45  import org.w3c.dom.DOMException;
46
47  /**
48   * This interface allows the DOM user to create a CSSStyleSheet
49   * outside the context of a document. There is no way to associate the new
50   * CSSStyleSheet with a document in DOM Level 2.
51   * 

See also the Document
52   * Object Model (DOM) Level 2 Style Specification.


53   * @since DOM Level 2
54   */
55  public interface DOMImplementationCSS extends DOMImplementation {
56      /**
57       * Creates a new CSSStyleSheet.
58       * @param title The advisory title. See also the section.
59       * @param media The comma-separated list of media associated with the
60       * new style sheet. See also the section.
61       * @return A new CSS style sheet.
62       * @exception DOMException
63       * SYNTAX_ERR: Raised if the specified media string value has a syntax
64       * error and is unparsable.
65       */
66      public CSSStyleSheet createCSSStyleSheet(String title,
67                                              String media)
68                                              throws DOMException;
69  }

```

Secuencia 378: org/w3c/dom/css/DocumentCSS.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.Element;
45 import org.w3c.dom.stylesheets.DocumentStyle;
46
47 /**
48 * This interface represents a document with a CSS view.
49 * <p> The <code>getOverrideStyle</code> method provides a mechanism through
50 * which a DOM author could effect immediate change to the style of an
51 * element without modifying the explicitly linked style sheets of a
52 * document or the inline style of elements in the style sheets. This style
```

```

53 * sheet comes after the author style sheet in the cascade algorithm and is
54 * called override style sheet. The override style sheet takes precedence
55 * over author style sheets. An "!important" declaration still takes
56 * precedence over a normal declaration. Override, author, and user style
57 * sheets all may contain "!important" declarations. User "!important" rules
58 * take precedence over both override and author "!important" rules, and
59 * override "!important" rules take precedence over author "!important"
60 * rules.
61 * <p> The expectation is that an instance of the <code>DocumentCSS</code>
62 * interface can be obtained by using binding-specific casting methods on an
63 * instance of the <code>Document</code> interface.
64 * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
    Object Model (DOM) Level 2 Style Specification</a>.
65 * @since DOM Level 2
66 */
67 public interface DocumentCSS extends DocumentStyle {
68     /**
69      * This method is used to retrieve the override style declaration for a
70      * specified element and a specified pseudo-element.
71      * @param elt The element whose style is to be modified. This parameter
72      * cannot be null.
73      * @param pseudoElt The pseudo-element or <code>null</code> if none.
74      * @return The override style declaration.
75      */
76     public CSSStyleDeclaration getOverrideStyle(Element elt,
77                                                  String pseudoElt);
78
79 }

```

### 38.19 ElementCSSInlineStyle.java

Secuencia 379: org/w3c/dom/css/ElementCSSInlineStyle.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```

22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  /**
45   * Inline style information attached to elements is exposed through the
46   * <code>style</code> attribute. This represents the contents of the STYLE
47   * attribute for HTML elements (or elements in other schemas or DTDs which
48   * use the STYLE attribute in the same way). The expectation is that an
49   * instance of the ElementCSSInlineStyle interface can be obtained by using
50   * binding-specific casting methods on an instance of the Element interface
51   * when the element supports inline CSS style informations.
52   * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
53   *   Object Model (DOM) Level 2 Style Specification</a>.
54   * @since DOM Level 2
55   */
56  public interface ElementCSSInlineStyle {
57      /**
58       * The style attribute.
59       */
60      public CSSStyleDeclaration getStyle();
61  }

```

## 38.20 RGBColor.java

Secuencia 380: org/w3c/dom/css/RGBColor.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 /**
45  * The RGBColor interface is used to represent any RGB color
46  * value. This interface reflects the values in the underlying style
47  * property. Hence, modifications made to the CSSPrimitiveValue
48  * objects modify the style property.
49  * <p> A specified RGB color is not clipped (even if the number is outside the
50  * range 0-255 or 0%-100%). A computed RGB color is clipped depending on the
51  * device.
52  * <p> Even if a style sheet can only contain an integer for a color value,
53  * the internal storage of this integer is a float, and this can be used as
54  * a float in the specified or the computed style.
55  * <p> A color percentage value can always be converted to a number and vice
56  * versa.
57  * <p> See also the Document
58  * Object Model (DOM) Level 2 Style Specification.
59  * @since DOM Level 2
60  */
61 public interface RGBColor {
```

```
61  /**
62   * This attribute is used for the red value of the RGB color.
63   */
64  public CSSPrimitiveValue getRed();
65
66  /**
67   * This attribute is used for the green value of the RGB color.
68   */
69  public CSSPrimitiveValue getGreen();
70
71  /**
72   * This attribute is used for the blue value of the RGB color.
73   */
74  public CSSPrimitiveValue getBlue();
75
76 }
```

### 38.21 Rect.java

Secuencia 381: [org.w3c/dom/css/Rect.java](http://org.w3c/dom/css/Rect.java)

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26   *
27   *
28   *
29   *
30   *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
```

```

34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.css;
43
44  /**
45   * The <code>Rect</code> interface is used to represent any rect value. This
46   * interface reflects the values in the underlying style property. Hence,
47   * modifications made to the <code>CSSPrimitiveValue</code> objects modify
48   * the style property.
49   * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
      Object Model (DOM) Level 2 Style Specification</a>.
50   * @since DOM Level 2
51   */
52  public interface Rect {
53      /**
54       * This attribute is used for the top of the rect.
55       */
56      public CSSPrimitiveValue getTop();
57
58      /**
59       * This attribute is used for the right of the rect.
60       */
61      public CSSPrimitiveValue getRight();
62
63      /**
64       * This attribute is used for the bottom of the rect.
65       */
66      public CSSPrimitiveValue getBottom();
67
68      /**
69       * This attribute is used for the left of the rect.
70       */
71      public CSSPrimitiveValue getLeft();
72  }
73

```

## 38.22 ViewCSS.java

Secuencia 382: org/w3c/dom/css/ViewCSS.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *

```



```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.css;
43
44 import org.w3c.dom.Element;
45 import org.w3c.dom.views.AbstractView;
46
47 /**
48  * This interface represents a CSS view. The getComputedStyle
49  * method provides a read only access to the computed values of an element.
50  * 

The expectation is that an instance of the ViewCSS
51  * interface can be obtained by using binding-specific casting methods on an
52  * instance of the AbstractView interface.
53  * 

Since a computed style is related to an Element node, if
54  * this element is removed from the document, the associated
55  * CSSStyleDeclaration and CSSValue related to
56  * this declaration are no longer valid.
57  * 

See also the Document
58  * Object Model (DOM) Level 2 Style Specification.


59  * @since DOM Level 2
60  */
61 public interface ViewCSS extends AbstractView {


```

```

61  /**
62   * This method is used to get the computed style as it is defined in [

```

## 39 Dom Events (org.w3c.dom.events package)

### 39.1 DocumentEvent.java

Secuencia 383: org.w3c/dom/events/DocumentEvent.java

```

1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software

```

```

35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.events;
43
44  import org.w3c.dom.DOMException;
45
46  /**
47   * The DocumentEvent interface provides a mechanism by which the
48   * user can create an Event of a type supported by the implementation. It is
49   * expected that the DocumentEvent interface will be
50   * implemented on the same object which implements the Document
51   * interface in an implementation which supports the Event model.
52   * 

See also the Document
53   * Object Model (DOM) Level 2 Events Specification.


54   * @since DOM Level 2
55   */
56  public interface DocumentEvent {
57      /**
58       *
59       * @param eventType The eventType parameter specifies the
60       * type of Event interface to be created. If the
61       * Event interface specified is supported by the
62       * implementation this method will return a new Event of
63       * the interface type requested. If the Event is to be
64       * dispatched via the dispatchEvent method the
65       * appropriate event init method must be called after creation in
66       * order to initialize the Event's values. As an example,
67       * a user wishing to synthesize some kind of UIEvent
68       * would call createEvent with the parameter "UIEvents".
69       * The initUIEvent method could then be called on the
70       * newly created UIEvent to set the specific type of
71       * UIEvent to be dispatched and set its context information. The
72       * createEvent method is used in creating
73       * Events when it is either inconvenient or unnecessary
74       * for the user to create an Event themselves. In cases
75       * where the implementation provided Event is
76       * insufficient, users may supply their own Event
77       * implementations for use with the dispatchEvent method.
78       * @return The newly created Event
79       * @exception DOMException
80       * NOT_SUPPORTED_ERR: Raised if the implementation does not support the
81       * type of Event interface requested
82       */
83      public Event createEvent(String eventType)
84                          throws DOMException;
85  }

```

## 39.2 Event.java

Secuencia 384: org/w3c/dom/events/Event.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.events;
43
44 /**
45 * The Event interface is used to provide contextual information
46 * about an event to the handler processing the event. An object which
47 * implements the Event interface is generally passed as the
48 * first parameter to an event handler. More specific context information is
49 * passed to event handlers by deriving additional interfaces from
50 * Event which contain information directly relating to the
```

```

51  * type of event they accompany. These derived interfaces are also
52  * implemented by the object passed to the event listener.
53  * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113'>Document
    Object Model (DOM) Level 2 Events Specification</a>.
54  * @since DOM Level 2
55  */
56  public interface Event {
57      // PhaseType
58      /**
59       * The current event phase is the capturing phase.
60       */
61      public static final short CAPTURING_PHASE          = 1;
62      /**
63       * The event is currently being evaluated at the target
64       * <code>EventTarget</code>.
65       */
66      public static final short AT_TARGET                = 2;
67      /**
68       * The current event phase is the bubbling phase.
69       */
70      public static final short BUBBLING_PHASE          = 3;
71
72      /**
73       * The name of the event (case-insensitive). The name must be an XML name.
74       */
75      public String getType();
76
77      /**
78       * Used to indicate the <code>EventTarget</code> to which the event was
79       * originally dispatched.
80       */
81      public EventTarget getTarget();
82
83      /**
84       * Used to indicate the <code>EventTarget</code> whose
85       * <code>EventListeners</code> are currently being processed. This is
86       * particularly useful during capturing and bubbling.
87       */
88      public EventTarget getCurrentTarget();
89
90      /**
91       * Used to indicate which phase of event flow is currently being
92       * evaluated.
93       */
94      public short getEventPhase();
95
96      /**
97       * Used to indicate whether or not an event is a bubbling event. If the
98       * event can bubble the value is true, else the value is false.
99       */
100     public boolean getBubbles();
101
102     /**

```

```
103     * Used to indicate whether or not an event can have its default action
104     * prevented. If the default action can be prevented the value is true,
105     * else the value is false.
106     */
107     public boolean getCancelable();
108
109     /**
110     * Used to specify the time (in milliseconds relative to the epoch) at
111     * which the event was created. Due to the fact that some systems may
112     * not provide this information the value of timeStamp may
113     * be not available for all events. When not available, a value of 0
114     * will be returned. Examples of epoch time are the time of the system
115     * start or 0:0:0 UTC 1st January 1970.
116     */
117     public long getTimeStamp();
118
119     /**
120     * The stopPropagation method is used prevent further
121     * propagation of an event during event flow. If this method is called
122     * by any EventListener the event will cease propagating
123     * through the tree. The event will complete dispatch to all listeners
124     * on the current EventTarget before event flow stops. This
125     * method may be used during any stage of event flow.
126     */
127     public void stopPropagation();
128
129     /**
130     * If an event is cancelable, the preventDefault method is
131     * used to signify that the event is to be canceled, meaning any default
132     * action normally taken by the implementation as a result of the event
133     * will not occur. If, during any stage of event flow, the
134     * preventDefault method is called the event is canceled.
135     * Any default action associated with the event will not occur. Calling
136     * this method for a non-cancelable event has no effect. Once
137     * preventDefault has been called it will remain in effect
138     * throughout the remainder of the event's propagation. This method may
139     * be used during any stage of event flow.
140     */
141     public void preventDefault();
142
143     /**
144     * The initEvent method is used to initialize the value of an
145     * Event created through the DocumentEvent
146     * interface. This method may only be called before the
147     * Event has been dispatched via the
148     * dispatchEvent method, though it may be called multiple
149     * times during that phase if necessary. If called multiple times the
150     * final invocation takes precedence. If called from a subclass of
151     * Event interface only the values specified in the
152     * initEvent method are modified, all other attributes are
153     * left unchanged.
154     * @param eventTypeArg Specifies the event type. This type may be any
155     *     event type currently defined in this specification or a new event
```

```

156     * type.. The string must be an XML name. Any new event type must not
157     * begin with any upper, lower, or mixed case version of the string
158     * "DOM". This prefix is reserved for future DOM event sets. It is
159     * also strongly recommended that third parties adding their own
160     * events use their own prefix to avoid confusion and lessen the
161     * probability of conflicts with other new events.
162     * @param canBubbleArg Specifies whether or not the event can bubble.
163     * @param cancelableArg Specifies whether or not the event's default
164     *   action can be prevented.
165     */
166     public void initEvent(String eventTypeArg,
167                          boolean canBubbleArg,
168                          boolean cancelableArg);
169
170 }

```

### 39.3 EventException.java

Secuencia 385: org/w3c/dom/events/EventException.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software

```

```

35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.events;
43
44  /**
45   * Event operations may throw an EventException as specified in
46   * their method descriptions.
47   * 

See also the Document Object Model \(DOM\) Level 2 Events Specification.


48   * @since DOM Level 2
49   */
50  public class EventException extends RuntimeException {
51      public EventException(short code, String message) {
52          super(message);
53          this.code = code;
54      }
55      public short    code;
56      // EventExceptionCode
57      /**
58       * If the Event's type was not specified by initializing the
59       * event before the method was called. Specification of the Event's type
60       * as null or an empty string will also trigger this
61       * exception.
62       */
63      public static final short UNSPECIFIED_EVENT_TYPE_ERR = 0;
64  }
65  }

```

### 39.4 EventListener.java

Secuencia 386: org/w3c/dom/events/EventListener.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```



```

18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.events;
43
44  /**
45   * The EventListener interface is the primary method for
46   * handling events. Users implement the EventListener interface
47   * and register their listener on an EventTarget using the
48   * AddEventListener method. The users should also remove their
49   * EventListener from its EventTarget after they
50   * have completed using the listener.
51   * 

When a Node is copied using the cloneNode
52   * method the EventListeners attached to the source
53   * Node are not attached to the copied Node. If
54   * the user wishes the same EventListeners to be added to the
55   * newly created copy the user must add them manually.
56   * 

See also the http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113 Document
57   * Object Model (DOM) Level 2 Events Specification.
58   * @since DOM Level 2
59   */
60  public interface EventListener {
61      /**
62       * This method is called whenever an event occurs of the type for which
63       * the EventListener interface was registered.
64       * @param evt The Event contains contextual information
65       * about the event. It also contains the stopPropagation
66       * and preventDefault methods which are used in
67       * determining the event's flow and default action.
68       */
69      public void handleEvent(Event evt);


```

70 }

### 39.5 EventTarget.java

Secuencia 387: org/w3c/dom/events/EventTarget.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.events;
43
44 /**
45 * The EventTarget interface is implemented by all
46 * Nodes in an implementation which supports the DOM Event
47 * Model. Therefore, this interface can be obtained by using
48 * binding-specific casting methods on an instance of the Node
```

```

49  * interface. The interface allows registration and removal of
50  * EventListeners on an EventTarget and dispatch
51  * of events to that EventTarget.
52  * 

See also the Document
    Object Model (DOM) Level 2 Events Specification.
53  * @since DOM Level 2
54  */
55  public interface EventTarget {
56      /**
57       * This method allows the registration of event listeners on the event
58       * target. If an EventListener is added to an
59       * EventTarget while it is processing an event, it will not
60       * be triggered by the current actions but may be triggered during a
61       * later stage of event flow, such as the bubbling phase.
62       *   
 If multiple identical EventListeners are registered
63       * on the same EventTarget with the same parameters the
64       * duplicate instances are discarded. They do not cause the
65       * EventListener to be called twice and since they are
66       * discarded they do not need to be removed with the
67       * removeEventListener method.
68       * @param type The event type for which the user is registering
69       * @param listener The listener parameter takes an interface
70       * implemented by the user which contains the methods to be called
71       * when the event occurs.
72       * @param useCapture If true, useCapture indicates that the
73       * user wishes to initiate capture. After initiating capture, all
74       * events of the specified type will be dispatched to the registered
75       * EventListener before being dispatched to any
76       * EventTargets beneath them in the tree. Events which
77       * are bubbling upward through the tree will not trigger an
78       * EventListener designated to use capture.
79       */
80      public void addEventListener(String type,
81                                  EventListener listener,
82                                  boolean useCapture);
83
84      /**
85       * This method allows the removal of event listeners from the event
86       * target. If an EventListener is removed from an
87       * EventTarget while it is processing an event, it will not
88       * be triggered by the current actions. EventListeners can
89       * never be invoked after being removed.
90       *   
 Calling removeEventListener with arguments which do
91       * not identify any currently registered EventListener on
92       * the EventTarget has no effect.
93       * @param type Specifies the event type of the EventListener
94       * being removed.
95       * @param listener The EventListener parameter indicates the
96       * EventListener to be removed.
97       * @param useCapture Specifies whether the EventListener
98       * being removed was registered as a capturing listener or not. If a
99       * listener was registered twice, one with capture and one without,
100      * each must be removed separately. Removal of a capturing listener


```

```

101     * does not affect a non-capturing version of the same listener, and
102     * vice versa.
103     */
104     public void removeEventListener(String type,
105                                     EventListener listener,
106                                     boolean useCapture);
107
108     /**
109     * This method allows the dispatch of events into the implementations
110     * event model. Events dispatched in this manner will have the same
111     * capturing and bubbling behavior as events dispatched directly by the
112     * implementation. The target of the event is the
113     * <code>EventTarget</code> on which <code>dispatchEvent</code> is
114     * called.
115     * @param evt Specifies the event type, behavior, and contextual
116     * information to be used in processing the event.
117     * @return The return value of <code>dispatchEvent</code> indicates
118     * whether any of the listeners which handled the event called
119     * <code>preventDefault</code>. If <code>preventDefault</code> was
120     * called the value is false, else the value is true.
121     * @exception EventException
122     * UNSPECIFIED_EVENT_TYPE_ERR: Raised if the <code>Event</code>'s type
123     * was not specified by initializing the event before
124     * <code>dispatchEvent</code> was called. Specification of the
125     * <code>Event</code>'s type as <code>null</code> or an empty string
126     * will also trigger this exception.
127     */
128     public boolean dispatchEvent(Event evt)
129                                     throws EventException;
130
131 }

```

### 39.6 MouseEvent.java

Secuencia 388: [org.w3c/dom/events/MouseEvent.java](http://org.w3c/dom/events/MouseEvent.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *

```

```
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.events;
43
44  import org.w3c.dom.views.AbstractView;
45
46  /**
47   * The MouseEvent interface provides specific contextual
48   * information associated with Mouse events.
49   * 

The detail attribute inherited from UIEvent
50   * indicates the number of times a mouse button has been pressed and
51   * released over the same screen location during a user action. The
52   * attribute value is 1 when the user begins this action and increments by 1
53   * for each full sequence of pressing and releasing. If the user moves the
54   * mouse between the mousedown and mouseup the value will be set to 0,
55   * indicating that no click is occurring.
56   * 

In the case of nested elements mouse events are always targeted at the
57   * most deeply nested element. Ancestors of the targeted element may use
58   * bubbling to obtain notification of mouse events which occur within its
59   * descendent elements.
60   * 

See also the Document
61   * Object Model (DOM) Level 2 Events Specification.


62   * @since DOM Level 2
63   */
64  public interface MouseEvent extends UIEvent {
65      /**
66       * The horizontal coordinate at which the event occurred relative to the
67       * origin of the screen coordinate system.
68       */
69      public int getScreenX();
70  }


```

```
71     * The vertical coordinate at which the event occurred relative to the
72     * origin of the screen coordinate system.
73     */
74     public int getScreenY();
75
76     /**
77     * The horizontal coordinate at which the event occurred relative to the
78     * DOM implementation's client area.
79     */
80     public int getClientX();
81
82     /**
83     * The vertical coordinate at which the event occurred relative to the DOM
84     * implementation's client area.
85     */
86     public int getClientY();
87
88     /**
89     * Used to indicate whether the 'ctrl' key was depressed during the firing
90     * of the event.
91     */
92     public boolean getCtrlKey();
93
94     /**
95     * Used to indicate whether the 'shift' key was depressed during the
96     * firing of the event.
97     */
98     public boolean getShiftKey();
99
100    /**
101    * Used to indicate whether the 'alt' key was depressed during the firing
102    * of the event. On some platforms this key may map to an alternative
103    * key name.
104    */
105    public boolean getAltKey();
106
107    /**
108    * Used to indicate whether the 'meta' key was depressed during the firing
109    * of the event. On some platforms this key may map to an alternative
110    * key name.
111    */
112    public boolean getMetaKey();
113
114    /**
115    * During mouse events caused by the depression or release of a mouse
116    * button, button is used to indicate which mouse button
117    * changed state. The values for button range from zero to
118    * indicate the left button of the mouse, one to indicate the middle
119    * button if present, and two to indicate the right button. For mice
120    * configured for left handed use in which the button actions are
121    * reversed the values are instead read from right to left.
122    */
123    public short getButton();
```

```
124
125 /**
126  * Used to identify a secondary <code>EventTarget</code> related to a UI
127  * event. Currently this attribute is used with the mouseover event to
128  * indicate the <code>EventTarget</code> which the pointing device
129  * exited and with the mouseout event to indicate the
130  * <code>EventTarget</code> which the pointing device entered.
131  */
132 public EventTarget getRelatedTarget();
133
134 /**
135  * The <code>initMouseEvent</code> method is used to initialize the value
136  * of a <code>MouseEvent</code> created through the
137  * <code>DocumentEvent</code> interface. This method may only be called
138  * before the <code>MouseEvent</code> has been dispatched via the
139  * <code>dispatchEvent</code> method, though it may be called multiple
140  * times during that phase if necessary. If called multiple times, the
141  * final invocation takes precedence.
142  * @param typeArg Specifies the event type.
143  * @param canBubbleArg Specifies whether or not the event can bubble.
144  * @param cancelableArg Specifies whether or not the event's default
145  *   action can be prevented.
146  * @param viewArg Specifies the <code>Event</code>'s
147  *   <code>AbstractView</code>.
148  * @param detailArg Specifies the <code>Event</code>'s mouse click count.
149  * @param screenXArg Specifies the <code>Event</code>'s screen x
150  *   coordinate
151  * @param screenYArg Specifies the <code>Event</code>'s screen y
152  *   coordinate
153  * @param clientXArg Specifies the <code>Event</code>'s client x
154  *   coordinate
155  * @param clientYArg Specifies the <code>Event</code>'s client y
156  *   coordinate
157  * @param ctrlKeyArg Specifies whether or not control key was depressed
158  *   during the <code>Event</code>.
159  * @param altKeyArg Specifies whether or not alt key was depressed during
160  *   the <code>Event</code>.
161  * @param shiftKeyArg Specifies whether or not shift key was depressed
162  *   during the <code>Event</code>.
163  * @param metaKeyArg Specifies whether or not meta key was depressed
164  *   during the <code>Event</code>.
165  * @param buttonArg Specifies the <code>Event</code>'s mouse button.
166  * @param relatedTargetArg Specifies the <code>Event</code>'s related
167  *   <code>EventTarget</code>.
168  */
169 public void initMouseEvent(String typeArg,
170                           boolean canBubbleArg,
171                           boolean cancelableArg,
172                           AbstractView viewArg,
173                           int detailArg,
174                           int screenXArg,
175                           int screenYArg,
176                           int clientXArg,
```

```
177         int clientYArg,
178         boolean ctrlKeyArg,
179         boolean altKeyArg,
180         boolean shiftKeyArg,
181         boolean metaKeyArg,
182         short buttonArg,
183         EventTarget relatedTargetArg);
184
185 }
```

### 39.7 MutationEvent.java

Secuencia 389: org/w3c/dom/events/MutationEvent.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
```



```
41
42 package org.w3c.dom.events;
43
44 import org.w3c.dom.Node;
45
46 /**
47  * The MutationEvent interface provides specific contextual
48  * information associated with Mutation events.
49  * 

See also the Document
      Object Model (DOM) Level 2 Events Specification

.
50  * @since DOM Level 2
51  */
52 public interface MutationEvent extends Event {
53     // attrChangeType
54     /**
55      * The Attr was modified in place.
56      */
57     public static final short MODIFICATION          = 1;
58     /**
59      * The Attr was just added.
60      */
61     public static final short ADDITION              = 2;
62     /**
63      * The Attr was just removed.
64      */
65     public static final short REMOVAL               = 3;
66
67     /**
68      * relatedNode is used to identify a secondary node related
69      * to a mutation event. For example, if a mutation event is dispatched
70      * to a node indicating that its parent has changed, the
71      * relatedNode is the changed parent. If an event is
72      * instead dispatched to a subtree indicating a node was changed within
73      * it, the relatedNode is the changed node. In the case of
74      * the DOMAttrModified event it indicates the Attr node
75      * which was modified, added, or removed.
76      */
77     public Node getRelatedNode();
78
79     /**
80      * prevValue indicates the previous value of the
81      * Attr node in DOMAttrModified events, and of the
82      * CharacterData node in DOMCharacterDataModified events.
83      */
84     public String getPrevValue();
85
86     /**
87      * newValue indicates the new value of the Attr
88      * node in DOMAttrModified events, and of the CharacterData
89      * node in DOMCharacterDataModified events.
90      */
91     public String getNewValue();
92 }
```

```

93  /**
94   * <code>attrName</code> indicates the name of the changed
95   * <code>Attr</code> node in a DOMAttrModified event.
96   */
97  public String getAttrName();
98
99  /**
100   * <code>attrChange</code> indicates the type of change which triggered
101   * the DOMAttrModified event. The values can be <code>MODIFICATION</code>
102   * , <code>ADDITION</code>, or <code>REMOVAL</code>.
103   */
104  public short getAttrChange();
105
106  /**
107   * The <code>initMutationEvent</code> method is used to initialize the
108   * value of a <code>MutationEvent</code> created through the
109   * <code>DocumentEvent</code> interface. This method may only be called
110   * before the <code>MutationEvent</code> has been dispatched via the
111   * <code>dispatchEvent</code> method, though it may be called multiple
112   * times during that phase if necessary. If called multiple times, the
113   * final invocation takes precedence.
114   * @param typeArg Specifies the event type.
115   * @param canBubbleArg Specifies whether or not the event can bubble.
116   * @param cancelableArg Specifies whether or not the event's default
117   *   action can be prevented.
118   * @param relatedNodeArg Specifies the <code>Event</code>'s related Node.
119   * @param prevValueArg Specifies the <code>Event</code>'s
120   *   <code>prevValue</code> attribute. This value may be null.
121   * @param newValueArg Specifies the <code>Event</code>'s
122   *   <code>newValue</code> attribute. This value may be null.
123   * @param attrNameArg Specifies the <code>Event</code>'s
124   *   <code>attrName</code> attribute. This value may be null.
125   * @param attrChangeArg Specifies the <code>Event</code>'s
126   *   <code>attrChange</code> attribute
127   */
128  public void initMutationEvent(String typeArg,
129                               boolean canBubbleArg,
130                               boolean cancelableArg,
131                               Node relatedNodeArg,
132                               String prevValueArg,
133                               String newValueArg,
134                               String attrNameArg,
135                               short attrChangeArg);
136
137 }

```

## 39.8 UIEvent.java

Secuencia 390: org/w3c/dom/events/UIEvent.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *

```

```
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.events;
43
44 import org.w3c.dom.views.AbstractView;
45
46 /**
47  * The UIEvent interface provides specific contextual information
48  * associated with User Interface events.
49  * 

See also the Document
49  * Object Model (DOM) Level 2 Events Specification.


50  * @since DOM Level 2
51  */
52 public interface UIEvent extends Event {
53     /**
54      * The view attribute identifies the AbstractView
55      * from which the event was generated.
56      */
57 }
```

```

57     public AbstractView getView();
58
59     /**
60      * Specifies some detail information about the <code>Event</code>,
61      * depending on the type of event.
62      */
63     public int getDetail();
64
65     /**
66      * The <code>initUIEvent</code> method is used to initialize the value of
67      * a <code>UIEvent</code> created through the <code>DocumentEvent</code>
68      * interface. This method may only be called before the
69      * <code>UIEvent</code> has been dispatched via the
70      * <code>dispatchEvent</code> method, though it may be called multiple
71      * times during that phase if necessary. If called multiple times, the
72      * final invocation takes precedence.
73      * @param typeArg Specifies the event type.
74      * @param canBubbleArg Specifies whether or not the event can bubble.
75      * @param cancelableArg Specifies whether or not the event's default
76      *       action can be prevented.
77      * @param viewArg Specifies the <code>Event</code>'s
78      *       <code>AbstractView</code>.
79      * @param detailArg Specifies the <code>Event</code>'s detail.
80      */
81     public void initUIEvent(String typeArg,
82                             boolean canBubbleArg,
83                             boolean cancelableArg,
84                             AbstractView viewArg,
85                             int detailArg);
86
87 }

```

## 40 Dom Html (org.w3c.dom.html package)

### 40.1 HTMLAnchorElement.java

Secuencia 391: org.w3c.dom.html/HTMLAnchorElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *

```

```
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * The anchor element. See the A element definition in HTML 4.0.
46   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47   *   Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLAnchorElement extends HTMLElement {
50      /**
51       * A single character access key to give access to the form control. See
52       * the accesskey attribute definition in HTML 4.0.
53       */
54      public String getAccessKey();
55      public void setAccessKey(String accessKey);
56
57      /**
58       * The character encoding of the linked resource. See the charset
59       * attribute definition in HTML 4.0.
60       */
61      public String getCharset();
62      public void setCharset(String charset);
63
64      /**
65       * Comma-separated list of lengths, defining an active region geometry.
66       * See also <code>shape</code> for the shape of the region. See the
67       * coords attribute definition in HTML 4.0.
68       */
69      public String getCoords();
```

```
69     public void setCoords(String coords);
70
71     /**
72      * The URI of the linked resource. See the href attribute definition in
73      * HTML 4.0.
74      */
75     public String getHref();
76     public void setHref(String href);
77
78     /**
79      * Language code of the linked resource. See the hreflang attribute
80      * definition in HTML 4.0.
81      */
82     public String getHreflang();
83     public void setHreflang(String hreflang);
84
85     /**
86      * Anchor name. See the name attribute definition in HTML 4.0.
87      */
88     public String getName();
89     public void setName(String name);
90
91     /**
92      * Forward link type. See the rel attribute definition in HTML 4.0.
93      */
94     public String getRel();
95     public void setRel(String rel);
96
97     /**
98      * Reverse link type. See the rev attribute definition in HTML 4.0.
99      */
100    public String getRev();
101    public void setRev(String rev);
102
103    /**
104     * The shape of the active area. The coordinates are given by
105     * <code>coords</code> . See the shape attribute definition in HTML 4.0.
106     */
107    public String getShape();
108    public void setShape(String shape);
109
110    /**
111     * Index that represents the element's position in the tabbing order. See
112     * the tabindex attribute definition in HTML 4.0.
113     */
114    public int getTabIndex();
115    public void setTabIndex(int tabIndex);
116
117    /**
118     * Frame to render the resource in. See the target attribute definition
119     * in HTML 4.0.
120     */
121    public String getTarget();
```

```
122     public void setTarget(String target);
123
124     /**
125      * Advisory content type. See the type attribute definition in HTML 4.0.
126      */
127     public String getType();
128     public void setType(String type);
129
130     /**
131      * Removes keyboard focus from this element.
132      */
133     public void blur();
134
135     /**
136      * Gives keyboard focus to this element.
137      */
138     public void focus();
139
140 }
```

## 40.2 HTMLAppletElement.java

Secuencia 392: org/w3c/dom/html/HTMLAppletElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
```

```
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * An embedded Java applet. See the APPLET element definition in HTML 4.0.
46  * This element is deprecated in HTML 4.0.
47  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48  *   Model (DOM) Level 2 Specification</a>.
49  */
49 public interface HTMLAppletElement extends HTMLElement {
50     /**
51      * Aligns this object (vertically or horizontally) with respect to its
52      * surrounding text. See the align attribute definition in HTML 4.0.
53      * This attribute is deprecated in HTML 4.0.
54      */
55     public String getAlign();
56     public void setAlign(String align);
57
58     /**
59      * Alternate text for user agents not rendering the normal content of
60      * this element. See the alt attribute definition in HTML 4.0. This
61      * attribute is deprecated in HTML 4.0.
62      */
63     public String getAlt();
64     public void setAlt(String alt);
65
66     /**
67      * Comma-separated archive list. See the archive attribute definition in
68      * HTML 4.0. This attribute is deprecated in HTML 4.0.
69      */
70     public String getArchive();
71     public void setArchive(String archive);
72
73     /**
74      * Applet class file. See the code attribute definition in HTML 4.0.
75      * This attribute is deprecated in HTML 4.0.
76      */
77     public String getCode();
78     public void setCode(String code);
79
80     /**
81      * Optional base URI for applet. See the codebase attribute definition
82      * in HTML 4.0. This attribute is deprecated in HTML 4.0.
```



```
83     */
84     public String getCodeBase();
85     public void setCodeBase(String codeBase);
86
87     /**
88      * Override height. See the height attribute definition in HTML 4.0.
89      * This attribute is deprecated in HTML 4.0.
90      */
91     public String getHeight();
92     public void setHeight(String height);
93
94     /**
95      * Horizontal space to the left and right of this image, applet, or
96      * object. See the hspace attribute definition in HTML 4.0. This
97      * attribute is deprecated in HTML 4.0.
98      */
99     public String getHspace();
100    public void setHspace(String hspace);
101
102    /**
103     * The name of the applet. See the name attribute definition in HTML
104     * 4.0. This attribute is deprecated in HTML 4.0.
105     */
106    public String getName();
107    public void setName(String name);
108
109    /**
110     * Serialized applet file. See the object attribute definition in HTML
111     * 4.0. This attribute is deprecated in HTML 4.0.
112     */
113    public String getObject();
114    public void setObject(String object);
115
116    /**
117     * Vertical space above and below this image, applet, or object. See the
118     * vspace attribute definition in HTML 4.0. This attribute is deprecated
119     * in HTML 4.0.
120     */
121    public String getVspace();
122    public void setVspace(String vspace);
123
124    /**
125     * Override width. See the width attribute definition in HTML 4.0. This
126     * attribute is deprecated in HTML 4.0.
127     */
128    public String getWidth();
129    public void setWidth(String width);
130
131 }
```

### 40.3 HTMLAreaElement.java

Secuencia 393: [org.w3c/dom/html/HTMLAreaElement.java](http://org.w3c/dom/html/HTMLAreaElement.java)

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45 * Client-side image map area definition. See the AREA element definition in
46 * HTML 4.0.
47 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48 *   Model (DOM) Level 2 Specification</a>.
49 */
50 public interface HTMLAreaElement extends HTMLElement {
51     /**
52      * A single character access key to give access to the form control. See
53      * the accesskey attribute definition in HTML 4.0.
```

```
53     */
54     public String getAccessKey();
55     public void setAccessKey(String accessKey);
56
57     /**
58      * Alternate text for user agents not rendering the normal content of
59      * this element. See the alt attribute definition in HTML 4.0.
60      */
61     public String getAlt();
62     public void setAlt(String alt);
63
64     /**
65      * Comma-separated list of lengths, defining an active region geometry.
66      * See also <code>shape</code> for the shape of the region. See the
67      * coords attribute definition in HTML 4.0.
68      */
69     public String getCoords();
70     public void setCoords(String coords);
71
72     /**
73      * The URI of the linked resource. See the href attribute definition in
74      * HTML 4.0.
75      */
76     public String getHref();
77     public void setHref(String href);
78
79     /**
80      * Specifies that this area is inactive, i.e., has no associated action.
81      * See the nohref attribute definition in HTML 4.0.
82      */
83     public boolean getNoHref();
84     public void setNoHref(boolean noHref);
85
86     /**
87      * The shape of the active area. The coordinates are given by
88      * <code>coords</code> . See the shape attribute definition in HTML 4.0.
89      */
90     public String getShape();
91     public void setShape(String shape);
92
93     /**
94      * Index that represents the element's position in the tabbing order. See
95      * the tabindex attribute definition in HTML 4.0.
96      */
97     public int getTabIndex();
98     public void setTabIndex(int tabIndex);
99
100     /**
101      * Frame to render the resource in. See the target attribute definition
102      * in HTML 4.0.
103      */
104     public String getTarget();
105     public void setTarget(String target);
```

```
106
107 }
```

#### 40.4 HTMLBRElement.java

Secuencia 394: org/w3c/dom/html/HTMLBRElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45 * Force a line break. See the BR element definition in HTML 4.0.
46 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
   Model (DOM) Level 2 Specification</a>.
```

```
47  */
48  public interface HTMLBRElement extends HTMLElement {
49      /**
50       * Control flow of text around floats. See the clear attribute definition
51       * in HTML 4.0. This attribute is deprecated in HTML 4.0.
52       */
53      public String getClear();
54      public void setClear(String clear);
55  }
56 }
```

## 40.5 HTMLElement.java

Secuencia 395: org/w3c/dom/html/HTMLElement.java

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
```

```
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Document base URI. See the BASE element definition in HTML 4.0.
46   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47   *   Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLBaseElement extends HTMLElement {
50      /**
51       * The base URI. See the href attribute definition in HTML 4.0.
52       */
53      public String getHref();
54      public void setHref(String href);
55
56      /**
57       * The default target frame. See the target attribute definition in HTML
58       * 4.0.
59       */
60      public String getTarget();
61      public void setTarget(String target);
62  }
```

## 40.6 HTMLBaseFontElement.java

Secuencia 396: org/w3c/dom/html/HTMLBaseFontElement.java

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
```

```

26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Base font. See the BASEFONT element definition in HTML 4.0. This element
46   * is deprecated in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLBaseFontElement extends HTMLElement {
50      /**
51       * Font color. See the color attribute definition in HTML 4.0. This
52       * attribute is deprecated in HTML 4.0.
53       */
54      public String getColor();
55      public void setColor(String color);
56
57      /**
58       * Font face identifier. See the face attribute definition in HTML 4.0.
59       * This attribute is deprecated in HTML 4.0.
60       */
61      public String getFace();
62      public void setFace(String face);
63
64      /**
65       * Font size. See the size attribute definition in HTML 4.0. This
66       * attribute is deprecated in HTML 4.0.
67       */
68      public String getSize();
69      public void setSize(String size);
70
71  }

```

## 40.7 HTMLBodyElement.java

Secuencia 397: org/w3c/dom/html/HTMLBodyElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

```

```
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * The HTML document body. This element is always present in the DOM API,
46  * even if the tags are not present in the source document. See the BODY
47  * element definition in HTML 4.0.
48  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
49  * Model (DOM) Level 2 Specification</a>.
50  */
51 public interface HTMLBodyElement extends HTMLElement {
52     /**
53      * Color of active links (after mouse-button down, but before
54      * mouse-button up). See the alink attribute definition in HTML 4.0.
55      * This attribute is deprecated in HTML 4.0.
```



```

55     */
56     public String getALink();
57     public void setALink(String aLink);
58
59     /**
60      * URI of the background texture tile image. See the background
61      * attribute definition in HTML 4.0. This attribute is deprecated in HTML
62      * 4.0.
63      */
64     public String getBackground();
65     public void setBackground(String background);
66
67     /**
68      * Document background color. See the bgcolor attribute definition in
69      * HTML 4.0. This attribute is deprecated in HTML 4.0.
70      */
71     public String getBgColor();
72     public void setBgColor(String bgColor);
73
74     /**
75      * Color of links that are not active and unvisited. See the link
76      * attribute definition in HTML 4.0. This attribute is deprecated in HTML
77      * 4.0.
78      */
79     public String getLink();
80     public void setLink(String link);
81
82     /**
83      * Document text color. See the text attribute definition in HTML 4.0.
84      * This attribute is deprecated in HTML 4.0.
85      */
86     public String getText();
87     public void setText(String text);
88
89     /**
90      * Color of links that have been visited by the user. See the vlink
91      * attribute definition in HTML 4.0. This attribute is deprecated in HTML
92      * 4.0.
93      */
94     public String getVLink();
95     public void setVLink(String vLink);
96
97 }

```

## 40.8 HTMLButtonElement.java

Secuencia 398: org/w3c/dom/html/HTMLButtonElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Push button. See the BUTTON element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLButtonElement extends HTMLElement {
50     /**
51      * Returns the <code>FORM</code> element containing this control. Returns
52      * <code>null</code> if this control is not within the context of a form.
53      */
54     public HTMLFormElement getForm();
55
56     /**
57      * A single character access key to give access to the form control. See
58      * the accesskey attribute definition in HTML 4.0.
59      */
59 }
```

```

59     public String getAccessKey();
60     public void setAccessKey(String accessKey);
61
62     /**
63      * The control is unavailable in this context. See the disabled
64      * attribute definition in HTML 4.0.
65      */
66     public boolean getDisabled();
67     public void setDisabled(boolean disabled);
68
69     /**
70      * Form control or object name when submitted with a form. See the name
71      * attribute definition in HTML 4.0.
72      */
73     public String getName();
74     public void setName(String name);
75
76     /**
77      * Index that represents the element's position in the tabbing order. See
78      * the tabIndex attribute definition in HTML 4.0.
79      */
80     public int getTabIndex();
81     public void setTabIndex(int tabIndex);
82
83     /**
84      * The type of button. See the type attribute definition in HTML 4.0.
85      */
86     public String getType();
87
88     /**
89      * The current form control value. See the value attribute definition in
90      * HTML 4.0.
91      */
92     public String getValue();
93     public void setValue(String value);
94
95 }

```

## 40.9 HTMLCollection.java

Secuencia 399: org/w3c/dom/html/HTMLCollection.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```

```
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  import org.w3c.dom.Node;
45
46  /**
47   * An HTMLCollection is a list of nodes. An individual node may
48   * be accessed by either ordinal index or the node's name or
49   * id attributes. Note: Collections in the HTML DOM are assumed
50   * to be live meaning that they are automatically updated when the
51   * underlying document is changed.
52   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
53   * Model (DOM) Level 2 Specification</a>.
54   */
55  public interface HTMLCollection {
56      /**
57       * This attribute specifies the length or size of the list.
58       */
59      public int getLength();
60
61      /**
62       * This method retrieves a node specified by ordinal index. Nodes are
63       * numbered in tree order (depth-first traversal order).
64       * @param index The index of the node to be fetched. The index origin is
65       * 0.
```

```

65     * @return The <code>Node</code> at the corresponding position upon
66     * success. A value of <code>null</code> is returned if the index is
67     * out of range.
68     */
69     public Node item(int index);
70
71     /**
72     * This method retrieves a <code>Node</code> using a name. It first
73     * searches for a <code>Node</code> with a matching <code>id</code>
74     * attribute. If it doesn't find one, it then searches for a
75     * <code>Node</code> with a matching <code>name</code> attribute, but
76     * only on those elements that are allowed a name attribute.
77     * @param name The name of the <code>Node</code> to be fetched.
78     * @return The <code>Node</code> with a <code>name</code> or
79     * <code>id</code> attribute whose value corresponds to the specified
80     * string. Upon failure (e.g., no node with this name exists), returns
81     * <code>null</code> .
82     */
83     public Node namedItem(String name);
84
85 }

```

## 40.10 HTMLDListElement.java

Secuencia 400: [org.w3c/dom/html/HTMLDListElement.java](http://org.w3c/dom/html/HTMLDListElement.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *

```

```

29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Definition list. See the DL element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  * Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLDListElement extends HTMLElement {
50     /**
51      * Reduce spacing between list items. See the compact attribute
52      * definition in HTML 4.0. This attribute is deprecated in HTML 4.0.
53      */
54     public boolean getCompact();
55     public void setCompact(boolean compact);
56 }

```

## 40.11 HTMLDOMImplementation.java

Secuencia 401: org/w3c/dom/html/HTMLDOMImplementation.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```

```

21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  import org.w3c.dom.DOMImplementation;
45
46  /**
47   * The HTMLDOMImplementation interface extends the
48   * DOMImplementation interface with a method for creating an
49   * HTML document instance.
50   * @since DOM Level 2
51   */
52  public interface HTMLDOMImplementation extends DOMImplementation {
53      /**
54       * Creates an HTMLDocument object with the minimal tree made
55       * of the following elements: HTML , HEAD ,
56       * TITLE , and BODY .
57       * @param title The title of the document to be set as the content of the
58       * TITLE element, through a child Text node.
59       * @return A new HTMLDocument object.
60       */
61      public HTMLDocument createHTMLDocument(String title);
62
63  }

```

## 40.12 HTMLDirectoryElement.java

Secuencia 402: org/w3c/dom/html/HTMLDirectoryElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Directory list. See the DIR element definition in HTML 4.0. This element
46  * is deprecated in HTML 4.0.
47  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48  *   Model (DOM) Level 2 Specification</a>.
49  */
49 public interface HTMLDirectoryElement extends HTMLElement {
50     /**
51      * Reduce spacing between list items. See the compact attribute
52      * definition in HTML 4.0. This attribute is deprecated in HTML 4.0.
53      */
54     public boolean getCompact();
55     public void setCompact(boolean compact);
56
57 }
```



**40.13** HTMLDivElement.java

Secuencia 403: org/w3c/dom/html/HTMLDivElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45 * Generic block container. See the DIV element definition in HTML 4.0.
46 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47 *   Model (DOM) Level 2 Specification</a>.
48 */
49 public interface HTMLDivElement extends HTMLElement {
```

```
50     * Horizontal text alignment. See the align attribute definition in HTML
51     * 4.0. This attribute is deprecated in HTML 4.0.
52     */
53     public String getAlign();
54     public void setAlign(String align);
55
56 }
```

#### 40.14 HTMLDocument.java

Secuencia 404: org/w3c/dom/html/HTMLDocument.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
```

```
43
44 import org.w3c.dom.Document;
45 import org.w3c.dom.NodeList;
46
47 /**
48  * An HTMLDocument is the root of the HTML hierarchy and holds
49  * the entire content. Besides providing access to the hierarchy, it also
50  * provides some convenience methods for accessing certain sets of
51  * information from the document.
52  * <p> The following properties have been deprecated in favor of the
53  * corresponding ones for the BODY element: alinkColor background
54  * bgColor fgColor linkColor vlinkColor In DOM Level 2, the method
55  * getElementById is inherited from the Document
56  * interface where it was moved.
57  * <p>See also the .
58  \*/
59 public interface HTMLDocument extends Document {
60     /\*\*
61      \* The title of a document as specified by the TITLE element
62      \* in the head of the document.
63      \*/
64     public String getTitle\(\);
65     public void setTitle\(String title\);
66
67     /\*\*
68      \* Returns the URI of the page that linked to this page. The value is an
69      \* empty string if the user navigated to the page directly \(not through a
70      \* link, but, for example, via a bookmark\).
71      \*/
72     public String getReferrer\(\);
73
74     /\*\*
75      \* The domain name of the server that served the document, or
76      \* null if the server cannot be identified by a domain name.
77      \*/
78     public String getDomain\(\);
79
80     /\*\*
81      \* The complete URI of the document.
82      \*/
83     public String getURL\(\);
84
85     /\*\*
86      \* The element that contains the content for the document. In documents
87      \* with BODY contents, returns the BODY
88      \* element. In frameset documents, this returns the outermost
89      \* FRAMESET element.
90      \*/
91     public HTMLElement getBody\(\);
92     public void setBody\(HTMLElement body\);
93
94     /\*\*
```

```
95     * A collection of all the <code>IMG</code> elements in a document. The
96     * behavior is limited to <code>IMG</code> elements for backwards
97     * compatibility.
98     */
99     public HTMLCollection getImages();
100
101     /**
102     * A collection of all the <code>OBJECT</code> elements that include
103     * applets and <code>APPLET</code> ( deprecated ) elements in a document.
104     */
105     public HTMLCollection getApplets();
106
107     /**
108     * A collection of all <code>AREA</code> elements and anchor (
109     * <code>A</code> ) elements in a document with a value for the
110     * <code>href</code> attribute.
111     */
112     public HTMLCollection getLinks();
113
114     /**
115     * A collection of all the forms of a document.
116     */
117     public HTMLCollection getForms();
118
119     /**
120     * A collection of all the anchor ( <code>A</code> ) elements in a document
121     * with a value for the <code>name</code> attribute. Note. For reasons
122     * of backwards compatibility, the returned set of anchors only contains
123     * those anchors created with the <code>name</code> attribute, not those
124     * created with the <code>id</code> attribute.
125     */
126     public HTMLCollection getAnchors();
127
128     /**
129     * The cookies associated with this document. If there are none, the
130     * value is an empty string. Otherwise, the value is a string: a
131     * semicolon-delimited list of "name, value" pairs for all the cookies
132     * associated with the page. For example,
133     * <code>name=value;expires=date</code> .
134     */
135     public String getCookie();
136     public void setCookie(String cookie);
137
138     /**
139     * Note. This method and the ones following allow a user to add to or
140     * replace the structure model of a document using strings of unparsed
141     * HTML. At the time of writing alternate methods for providing similar
142     * functionality for both HTML and XML documents were being considered.
143     * The following methods may be deprecated at some point in the future in
144     * favor of a more general-purpose mechanism.
145     * <br> Open a document stream for writing. If a document exists in the
146     * target, this method clears it.
147     */
```

```

148     public void open();
149
150     /**
151      * Closes a document stream opened by <code>open()</code> and forces
152      * rendering.
153      */
154     public void close();
155
156     /**
157      * Write a string of text to a document stream opened by
158      * <code>open()</code> . The text is parsed into the document's structure
159      * model.
160      * @param text The string to be parsed into some structure in the
161      * document structure model.
162      */
163     public void write(String text);
164
165     /**
166      * Write a string of text followed by a newline character to a document
167      * stream opened by <code>open()</code> . The text is parsed into the
168      * document's structure model.
169      * @param text The string to be parsed into some structure in the
170      * document structure model.
171      */
172     public void writeln(String text);
173
174     /**
175      * Returns the (possibly empty) collection of elements whose
176      * <code>name</code> value is given by <code>elementName</code> .
177      * @param elementName The <code>name</code> attribute value for an
178      * element.
179      * @return The matching elements.
180      */
181     public NodeList getElementsByName(String elementName);
182
183 }

```

## 40.15 HTMLElement.java

Secuencia 405: org/w3c/dom/html/HTMLElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```

14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  import org.w3c.dom.Element;
45
46  /**
47   * All HTML element interfaces derive from this class. Elements that only
48   * expose the HTML core attributes are represented by the base
49   * HTMLElement interface. These elements are as follows: HEAD
50   * special: SUB, SUP, SPAN, BDO font: TT, I, B, U, S, STRIKE, BIG, SMALL
51   * phrase: EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ACRONYM, ABBR list:
52   * DD, DT NOFRAMES, NOSCRIPT ADDRESS, CENTER The style attribute
53   * of an HTML element is accessible through the
54   * ElementCSSInlineStyle interface which is defined in the .
55   * 

See also the http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510 Document Object
56   * Model (DOM) Level 2 Specification

.
57   */
58  public interface HTMLElement extends Element {
59      /**
60       * The element's identifier. See the id attribute definition in HTML 4.0.
61       */
62      public String getId();
63      public void setId(String id);
64
65      /**
66       * The element's advisory title. See the title attribute definition in

```

```
66     * HTML 4.0.
67     */
68     public String getTitle();
69     public void setTitle(String title);
70
71     /**
72     * Language code defined in RFC 1766. See the lang attribute definition
73     * in HTML 4.0.
74     */
75     public String getLang();
76     public void setLang(String lang);
77
78     /**
79     * Specifies the base direction of directionally neutral text and the
80     * directionality of tables. See the dir attribute definition in HTML
81     * 4.0.
82     */
83     public String getDir();
84     public void setDir(String dir);
85
86     /**
87     * The class attribute of the element. This attribute has been renamed
88     * due to conflicts with the "class" keyword exposed by many languages.
89     * See the class attribute definition in HTML 4.0.
90     */
91     public String getClassName();
92     public void setClassName(String className);
93
94 }
```

#### 40.16 HTMLFieldSetElement.java

Secuencia 406: org/w3c/dom/html/HTMLFieldSetElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
```

```

21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Organizes form controls into logical groups. See the FIELDSET element
46   * definition in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48   * Model (DOM) Level 2 Specification</a>.
49   */
49  public interface HTMLFieldSetElement extends HTMLElement {
50      /**
51       * Returns the <code>FORM</code> element containing this control. Returns
52       * <code>null</code> if this control is not within the context of a form.
53       */
54      public HTMLFormElement getForm();
55
56  }

```

## 40.17 HTMLFontElement.java

Secuencia 407: org/w3c/dom/html/HTMLFontElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *

```



```
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Local change to font. See the FONT element definition in HTML 4.0. This
46   * element is deprecated in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLFontElement extends HTMLElement {
50      /**
51       * Font color. See the color attribute definition in HTML 4.0. This
52       * attribute is deprecated in HTML 4.0.
53       */
54      public String getColor();
55      public void setColor(String color);
56
57      /**
58       * Font face identifier. See the face attribute definition in HTML 4.0.
59       * This attribute is deprecated in HTML 4.0.
60       */
61      public String getFace();
62      public void setFace(String face);
63
64      /**
```

```
65     * Font size. See the size attribute definition in HTML 4.0. This
66     * attribute is deprecated in HTML 4.0.
67     */
68     public String getSize();
69     public void setSize(String size);
70
71 }
```

## 40.18 HTMLFormElement.java

Secuencia 408: org/w3c/dom/html/HTMLFormElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
```

```
43
44 /**
45  * The <code>FORM</code> element encompasses behavior similar to a collection
46  * and an element. It provides direct access to the contained input elements
47  * as well as the attributes of the form element. See the FORM element
48  * definition in HTML 4.0.
49  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
50  */
51 public interface HTMLFormElement extends HTMLElement {
52     /**
53      * Returns a collection of all control elements in the form.
54      */
55     public HTMLCollection getElements();
56
57     /**
58      * The number of form controls in the form.
59      */
60     public int getLength();
61
62     /**
63      * Names the form.
64      */
65     public String getName();
66     public void setName(String name);
67
68     /**
69      * List of character sets supported by the server. See the
70      * accept-charset attribute definition in HTML 4.0.
71      */
72     public String getAcceptCharset();
73     public void setAcceptCharset(String acceptCharset);
74
75     /**
76      * Server-side form handler. See the action attribute definition in HTML
77      * 4.0.
78      */
79     public String getAction();
80     public void setAction(String action);
81
82     /**
83      * The content type of the submitted form, generally
84      * "application/x-www-form-urlencoded". See the enctype attribute
85      * definition in HTML 4.0.
86      */
87     public String getEnctype();
88     public void setEnctype(String enctype);
89
90     /**
91      * HTTP method used to submit form. See the method attribute definition
92      * in HTML 4.0.
93      */
94     public String getMethod();
```

```
95     public void setMethod(String method);
96
97     /**
98      * Frame to render the resource in. See the target attribute definition
99      * in HTML 4.0.
100     */
101     public String getTarget();
102     public void setTarget(String target);
103
104     /**
105      * Submits the form. It performs the same action as a submit button.
106     */
107     public void submit();
108
109     /**
110      * Restores a form element's default values. It performs the same action
111      * as a reset button.
112     */
113     public void reset();
114
115 }
```

#### 40.19 HTMLFrameElement.java

Secuencia 409: [org.w3c/dom/html/HTMLFrameElement.java](http://org.w3c/dom/html/HTMLFrameElement.java)

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
```

```
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42 package org.w3c.dom.html;
43
44 import org.w3c.dom.Document;
45
46 /**
47  * Create a frame. See the FRAME element definition in HTML 4.0.
48  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
49  * Model (DOM) Level 2 Specification</a>.
50  */
51 public interface HTMLFrameElement extends HTMLElement {
52     /**
53      * Request frame borders. See the frameborder attribute definition in
54      * HTML 4.0.
55      */
56     public String getFrameBorder();
57     public void setFrameBorder(String frameBorder);
58
59     /**
60      * URI designating a long description of this image or frame. See the
61      * longdesc attribute definition in HTML 4.0.
62      */
63     public String getLongDesc();
64     public void setLongDesc(String longDesc);
65
66     /**
67      * Frame margin height, in pixels. See the marginheight attribute
68      * definition in HTML 4.0.
69      */
70     public String getMarginHeight();
71     public void setMarginHeight(String marginHeight);
72
73     /**
74      * Frame margin width, in pixels. See the marginwidth attribute
75      * definition in HTML 4.0.
76      */
77     public String getMarginWidth();
78     public void setMarginWidth(String marginWidth);
79
80     /**
81      * The frame name (object of the <code>target</code> attribute). See the
```

```

81     * name attribute definition in HTML 4.0.
82     */
83     public String getName();
84     public void setName(String name);
85
86     /**
87     * When true, forbid user from resizing frame. See the noresize
88     * attribute definition in HTML 4.0.
89     */
90     public boolean getNoResize();
91     public void setNoResize(boolean noResize);
92
93     /**
94     * Specify whether or not the frame should have scrollbars. See the
95     * scrolling attribute definition in HTML 4.0.
96     */
97     public String getScrolling();
98     public void setScrolling(String scrolling);
99
100    /**
101    * A URI designating the initial frame contents. See the src attribute
102    * definition in HTML 4.0.
103    */
104    public String getSrc();
105    public void setSrc(String src);
106
107    /**
108    * The document this frame contains, if there is any and it is available,
109    * or <code>null</code> otherwise.
110    * @since DOM Level 2
111    */
112    public Document getContentDocument();
113
114 }

```

## 40.20 HTMLFrameSetElement.java

Secuencia 410: [org.w3c/dom/html/HTMLFrameSetElement.java](http://org.w3c/dom/html/HTMLFrameSetElement.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Create a grid of frames. See the FRAMESET element definition in HTML 4.0.
46   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47   *   Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLFrameSetElement extends HTMLElement {
50      /**
51       * The number of columns of frames in the frameset. See the cols
52       * attribute definition in HTML 4.0.
53       */
54      public String getCols();
55      public void setCols(String cols);
56
57      /**
58       * The number of rows of frames in the frameset. See the rows attribute
59       * definition in HTML 4.0.
60       */
61      public String getRows();
62      public void setRows(String rows);
63  }

```

## 40.21 HTMLHRElement.java

Secuencia 411: org/w3c/dom/html/HTMLHRElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45 * Create a horizontal rule. See the HR element definition in HTML 4.0.
46 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
47 */
48 public interface HTMLHRElement extends HTMLElement {
49     /**
50      * Align the rule on the page. See the align attribute definition in
51      * HTML 4.0. This attribute is deprecated in HTML 4.0.
52      */
```



```

53     public String getAlign();
54     public void setAlign(String align);
55
56     /**
57      * Indicates to the user agent that there should be no shading in the
58      * rendering of this element. See the noshade attribute definition in
59      * HTML 4.0. This attribute is deprecated in HTML 4.0.
60      */
61     public boolean getNoShade();
62     public void setNoShade(boolean noShade);
63
64     /**
65      * The height of the rule. See the size attribute definition in HTML
66      * 4.0. This attribute is deprecated in HTML 4.0.
67      */
68     public String getSize();
69     public void setSize(String size);
70
71     /**
72      * The width of the rule. See the width attribute definition in HTML
73      * 4.0. This attribute is deprecated in HTML 4.0.
74      */
75     public String getWidth();
76     public void setWidth(String width);
77
78 }

```

## 40.22 HTMLHeadElement.java

Secuencia 412: org/w3c/dom/html/HTMLHeadElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */

```

```

24
25 /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Document head information. See the HEAD element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49
50 public interface HTMLHeadElement extends HTMLElement {
51     /**
52      * URI designating a metadata profile. See the profile attribute
53      * definition in HTML 4.0.
54      */
55     public String getProfile();
56     public void setProfile(String profile);
57 }

```

## 40.23 HTMLHeadingElement.java

Secuencia 413: org/w3c/dom/html/HTMLHeadingElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * For the <code>H1</code> to <code>H6</code> elements. See the H1 element
46   * definition in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48   * Model (DOM) Level 2 Specification</a>.
49   */
49  public interface HTMLHeadingElement extends HTMLElement {
50      /**
51       * Horizontal text alignment. See the align attribute definition in HTML
52       * 4.0. This attribute is deprecated in HTML 4.0.
53       */
54      public String getAlign();
55      public void setAlign(String align);
56
57  }

```

## 40.24 HTMLHtmlElement.java

Secuencia 414: org/w3c/dom/html/HTMLHtmlElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Root of an HTML document. See the HTML element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLHtmlElement extends HTMLElement {
50     /**
51      * Version information about the document's DTD. See the version
52      * attribute definition in HTML 4.0. This attribute is deprecated in HTML
53      * 4.0.
54      */
55     public String getVersion();
56     public void setVersion(String version);
57 }
```

**40.25** HTMLIFrameElement.java

Secuencia 415: org/w3c/dom/html/HTMLIFrameElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 import org.w3c.dom.Document;
45
46 /**
47 * Inline subwindows. See the IFRAME element definition in HTML 4.0.
48 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
   Model (DOM) Level 2 Specification</a>.
49 */
```

```
50 public interface HTMLIFrameElement extends HTMLElement {
51     /**
52      * Aligns this object (vertically or horizontally) with respect to its
53      * surrounding text. See the align attribute definition in HTML 4.0.
54      * This attribute is deprecated in HTML 4.0.
55      */
56     public String getAlign();
57     public void setAlign(String align);
58
59     /**
60      * Request frame borders. See the frameborder attribute definition in
61      * HTML 4.0.
62      */
63     public String getFrameBorder();
64     public void setFrameBorder(String frameBorder);
65
66     /**
67      * Frame height. See the height attribute definition in HTML 4.0.
68      */
69     public String getHeight();
70     public void setHeight(String height);
71
72     /**
73      * URI designating a long description of this image or frame. See the
74      * longdesc attribute definition in HTML 4.0.
75      */
76     public String getLongDesc();
77     public void setLongDesc(String longDesc);
78
79     /**
80      * Frame margin height, in pixels. See the marginheight attribute
81      * definition in HTML 4.0.
82      */
83     public String getMarginHeight();
84     public void setMarginHeight(String marginHeight);
85
86     /**
87      * Frame margin width, in pixels. See the marginwidth attribute
88      * definition in HTML 4.0.
89      */
90     public String getMarginWidth();
91     public void setMarginWidth(String marginWidth);
92
93     /**
94      * The frame name (object of the <code>target</code> attribute). See the
95      * name attribute definition in HTML 4.0.
96      */
97     public String getName();
98     public void setName(String name);
99
100     /**
101      * Specify whether or not the frame should have scrollbars. See the
102      * scrolling attribute definition in HTML 4.0.
```

```

103     */
104     public String getScrolling();
105     public void setScrolling(String scrolling);
106
107     /**
108      * A URI designating the initial frame contents. See the src attribute
109      * definition in HTML 4.0.
110      */
111     public String getSrc();
112     public void setSrc(String src);
113
114     /**
115      * Frame width. See the width attribute definition in HTML 4.0.
116      */
117     public String getWidth();
118     public void setWidth(String width);
119
120     /**
121      * The document this frame contains, if there is any and it is available,
122      * or <code>null</code> otherwise.
123      * @since DOM Level 2
124      */
125     public Document getContentDocument();
126
127 }

```

## 40.26 HTMLImageElement.java

Secuencia 416: org/w3c/dom/html/HTMLImageElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24

```

```
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Embedded image. See the IMG element definition in HTML 4.0.
46   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47   *   Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLImageElement extends HTMLElement {
50      /**
51       * URI designating the source of this image, for low-resolution output.
52       */
53      public String getLowSrc();
54      public void setLowSrc(String lowSrc);
55
56      /**
57       * The name of the element (for backwards compatibility).
58       */
59      public String getName();
60      public void setName(String name);
61
62      /**
63       * Aligns this object (vertically or horizontally) with respect to its
64       * surrounding text. See the align attribute definition in HTML 4.0.
65       * This attribute is deprecated in HTML 4.0.
66       */
67      public String getAlign();
68      public void setAlign(String align);
69
70      /**
71       * Alternate text for user agents not rendering the normal content of
72       * this element. See the alt attribute definition in HTML 4.0.
73       */
74      public String getAlt();
75      public void setAlt(String alt);
76  }
```



```
77      * Width of border around image. See the border attribute definition in
78      * HTML 4.0. This attribute is deprecated in HTML 4.0.
79      */
80      public String getBorder();
81      public void setBorder(String border);
82
83      /**
84      * Override height. See the height attribute definition in HTML 4.0.
85      */
86      public String getHeight();
87      public void setHeight(String height);
88
89      /**
90      * Horizontal space to the left and right of this image. See the hspace
91      * attribute definition in HTML 4.0. This attribute is deprecated in HTML
92      * 4.0.
93      */
94      public String getHspace();
95      public void setHspace(String hspace);
96
97      /**
98      * Use server-side image map. See the ismap attribute definition in HTML
99      * 4.0.
100     */
101     public boolean getIsMap();
102     public void setIsMap(boolean isMap);
103
104     /**
105     * URI designating a long description of this image or frame. See the
106     * longdesc attribute definition in HTML 4.0.
107     */
108     public String getLongDesc();
109     public void setLongDesc(String longDesc);
110
111     /**
112     * URI designating the source of this image. See the src attribute
113     * definition in HTML 4.0.
114     */
115     public String getSrc();
116     public void setSrc(String src);
117
118     /**
119     * Use client-side image map. See the usemap attribute definition in
120     * HTML 4.0.
121     */
122     public String getUseMap();
123     public void setUseMap(String useMap);
124
125     /**
126     * Vertical space above and below this image. See the vspace attribute
127     * definition in HTML 4.0. This attribute is deprecated in HTML 4.0.
128     */
129     public String getVspace();
```

```
130     public void setVspace(String vspace);
131
132     /**
133      * Override width. See the width attribute definition in HTML 4.0.
134      */
135     public String getWidth();
136     public void setWidth(String width);
137
138 }
```

## 40.27 HTMLInputElement.java

Secuencia 417: org/w3c/dom/html/HTMLInputElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
```

```
41
42 package org.w3c.dom.html;
43
44 /**
45  * Form control. Note. Depending upon the environment in which the page is
46  * being viewed, the value property may be read-only for the file upload
47  * input type. For the "password" input type, the actual value returned may
48  * be masked to prevent unauthorized use. See the INPUT element definition
49  * in HTML 4.0.
50  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
51  */
52 public interface HTMLInputElement extends HTMLElement {
53     /**
54      * When the <code>type</code> attribute of the element has the value
55      * "Text", "File" or "Password", this represents the HTML value attribute
56      * of the element. The value of this attribute does not change if the
57      * contents of the corresponding form control, in an interactive user
58      * agent, changes. Changing this attribute, however, resets the contents
59      * of the form control. See the value attribute definition in HTML 4.0.
60      */
61     public String getDefaultValue();
62     public void setDefaultValue(String defaultValue);
63
64     /**
65      * When <code>type</code> has the value "Radio" or "Checkbox", this
66      * represents the HTML checked attribute of the element. The value of
67      * this attribute does not change if the state of the corresponding form
68      * control, in an interactive user agent, changes. Changes to this
69      * attribute, however, resets the state of the form control. See the
70      * checked attribute definition in HTML 4.0.
71      */
72     public boolean getDefaultChecked();
73     public void setDefaultChecked(boolean defaultChecked);
74
75     /**
76      * Returns the <code>FORM</code> element containing this control. Returns
77      * <code>null</code> if this control is not within the context of a form.
78      */
79     public HTMLFormElement getForm();
80
81     /**
82      * A comma-separated list of content types that a server processing this
83      * form will handle correctly. See the accept attribute definition in
84      * HTML 4.0.
85      */
86     public String getAccept();
87     public void setAccept(String accept);
88
89     /**
90      * A single character access key to give access to the form control. See
91      * the accesskey attribute definition in HTML 4.0.
92      */
93 }
```

```
93 public String getAccessKey();
94 public void setAccessKey(String accessKey);
95
96 /**
97  * Aligns this object (vertically or horizontally) with respect to its
98  * surrounding text. See the align attribute definition in HTML 4.0.
99  * This attribute is deprecated in HTML 4.0.
100 */
101 public String getAlign();
102 public void setAlign(String align);
103
104 /**
105  * Alternate text for user agents not rendering the normal content of
106  * this element. See the alt attribute definition in HTML 4.0.
107 */
108 public String getAlt();
109 public void setAlt(String alt);
110
111 /**
112  * When the <code>type</code> attribute of the element has the value
113  * "Radio" or "Checkbox", this represents the current state of the form
114  * control, in an interactive user agent. Changes to this attribute
115  * change the state of the form control, but do not change the value of
116  * the HTML value attribute of the element.
117 */
118 public boolean getChecked();
119 public void setChecked(boolean checked);
120
121 /**
122  * The control is unavailable in this context. See the disabled
123  * attribute definition in HTML 4.0.
124 */
125 public boolean getDisabled();
126 public void setDisabled(boolean disabled);
127
128 /**
129  * Maximum number of characters for text fields, when <code>type</code>
130  * has the value "Text" or "Password". See the maxlength attribute
131  * definition in HTML 4.0.
132 */
133 public int getMaxLength();
134 public void setMaxLength(int maxLength);
135
136 /**
137  * Form control or object name when submitted with a form. See the name
138  * attribute definition in HTML 4.0.
139 */
140 public String getName();
141 public void setName(String name);
142
143 /**
144  * This control is read-only. Relevant only when <code>type</code> has
145  * the value "Text" or "Password". See the readonly attribute definition
```

```
146     * in HTML 4.0.
147     */
148     public boolean getReadOnly();
149     public void setReadOnly(boolean readOnly);
150
151     /**
152     * Size information. The precise meaning is specific to each type of
153     * field. See the size attribute definition in HTML 4.0.
154     */
155     public String getSize();
156     public void setSize(String size);
157
158     /**
159     * When the <code>type</code> attribute has the value "Image", this
160     * attribute specifies the location of the image to be used to decorate
161     * the graphical submit button. See the src attribute definition in HTML
162     * 4.0.
163     */
164     public String getSrc();
165     public void setSrc(String src);
166
167     /**
168     * Index that represents the element's position in the tabbing order. See
169     * the tabindex attribute definition in HTML 4.0.
170     */
171     public int getTabIndex();
172     public void setTabIndex(int tabIndex);
173
174     /**
175     * The type of control created. See the type attribute definition in
176     * HTML 4.0.
177     */
178     public String getType();
179
180     /**
181     * Use client-side image map. See the usemap attribute definition in
182     * HTML 4.0.
183     */
184     public String getUseMap();
185     public void setUseMap(String useMap);
186
187     /**
188     * When the <code>type</code> attribute of the element has the value
189     * "Text", "File" or "Password", this represents the current contents of
190     * the corresponding form control, in an interactive user agent. Changing
191     * this attribute changes the contents of the form control, but does not
192     * change the value of the HTML value attribute of the element. When the
193     * <code>type</code> attribute of the element has the value "Button",
194     * "Hidden", "Submit", "Reset", "Image", "Checkbox" or "Radio", this
195     * represents the HTML value attribute of the element. See the value
196     * attribute definition in HTML 4.0.
197     */
198     public String getValue();
```

```
199     public void setValue(String value);
200
201     /**
202      * Removes keyboard focus from this element.
203      */
204     public void blur();
205
206     /**
207      * Gives keyboard focus to this element.
208      */
209     public void focus();
210
211     /**
212      * Select the contents of the text area. For <code>INPUT</code> elements
213      * whose <code>type</code> attribute has one of the following values:
214      * "Text", "File", or "Password".
215      */
216     public void select();
217
218     /**
219      * Simulate a mouse-click. For <code>INPUT</code> elements whose
220      * <code>type</code> attribute has one of the following values: "Button",
221      * "Checkbox", "Radio", "Reset", or "Submit".
222      */
223     public void click();
224
225 }
```

## 40.28 HTMLIsIndexElement.java

Secuencia 418: org/w3c/dom/html/HTMLIsIndexElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
```

```

23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * This element is used for single-line text input. See the ISINDEX element
46   * definition in HTML 4.0. This element is deprecated in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLIsIndexElement extends HTMLElement {
50      /**
51       * Returns the <code>FORM</code> element containing this control. Returns
52       * <code>null</code> if this control is not within the context of a form.
53       */
54      public HTMLFormElement getForm();
55
56      /**
57       * The prompt message. See the prompt attribute definition in HTML 4.0.
58       * This attribute is deprecated in HTML 4.0.
59       */
60      public String getPrompt();
61      public void setPrompt(String prompt);
62
63  }

```

## 40.29 HTMLLIElement.java

Secuencia 419: org/w3c/dom/html/HTMLLIElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *

```

```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * List item. See the LI element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLLIElement extends HTMLElement {
50     /**
51      * List item bullet style. See the type attribute definition in HTML
52      * 4.0. This attribute is deprecated in HTML 4.0.
53      */
54     public String getType();
55     public void setType(String type);
56
57     /**
58      * Reset sequence number when used in <code>OL</code> . See the value
59      * attribute definition in HTML 4.0. This attribute is deprecated in HTML
60      * 4.0.
```



```
60     */
61     public int getValue();
62     public void setValue(int value);
63
64 }
```

#### 40.30 HTMLLabelElement.java

Secuencia 420: org/w3c/dom/html/HTMLLabelElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
```

```

45  * Form field label text. See the LABEL element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
47  */
48  public interface HTMLLabelElement extends HTMLElement {
49      /**
50       * Returns the <code>FORM</code> element containing this control. Returns
51       * <code>null</code> if this control is not within the context of a form.
52       */
53      public HTMLFormElement getForm();
54
55      /**
56       * A single character access key to give access to the form control. See
57       * the accesskey attribute definition in HTML 4.0.
58       */
59      public String getAccessKey();
60      public void setAccessKey(String accessKey);
61
62      /**
63       * This attribute links this label with another form control by
64       * <code>id</code> attribute. See the for attribute definition in HTML
65       * 4.0.
66       */
67      public String getHtmlFor();
68      public void setHtmlFor(String htmlFor);
69
70  }

```

### 40.31 HTMLLegendElement.java

Secuencia 421: org/w3c/dom/html/HTMLLegendElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *

```

```

23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Provides a caption for a FIELDSET grouping. See the LEGEND
46   * element definition in HTML 4.0.
47   * 

See also the Document Object
48   * Model (DOM) Level 2 Specification.


49   */
49  public interface HTMLLegendElement extends HTMLElement {
50      /**
51       * Returns the FORM element containing this control. Returns
52       * null if this control is not within the context of a form.
53       */
54      public HTMLFormElement getForm();
55
56      /**
57       * A single character access key to give access to the form control. See
58       * the accesskey attribute definition in HTML 4.0.
59       */
60      public String getAccessKey();
61      public void setAccessKey(String accessKey);
62
63      /**
64       * Text alignment relative to FIELDSET. See the align
65       * attribute definition in HTML 4.0. This attribute is deprecated in HTML
66       * 4.0.
67       */
68      public String getAlign();
69      public void setAlign(String align);
70
71  }

```

Secuencia 422: org/w3c/dom/html/HTMLLinkElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45 * The <code>LINK</code> element specifies a link to an external resource,
46 * and defines this document's relationship to that resource (or vice versa).
47 * See the LINK element definition in HTML 4.0 (see also the
48 * <code>LinkStyle</code> interface in the module).
49 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
50 */
51 public interface HTMLLinkElement extends HTMLElement {
```

```
52  /**
53   * Enables/disables the link. This is currently only used for style sheet
54   * links, and may be used to activate or deactivate style sheets.
55   */
56  public boolean getDisabled();
57  public void setDisabled(boolean disabled);
58
59  /**
60   * The character encoding of the resource being linked to. See the
61   * charset attribute definition in HTML 4.0.
62   */
63  public String getCharset();
64  public void setCharset(String charset);
65
66  /**
67   * The URI of the linked resource. See the href attribute definition in
68   * HTML 4.0.
69   */
70  public String getHref();
71  public void setHref(String href);
72
73  /**
74   * Language code of the linked resource. See the hreflang attribute
75   * definition in HTML 4.0.
76   */
77  public String getHreflang();
78  public void setHreflang(String hreflang);
79
80  /**
81   * Designed for use with one or more target media. See the media
82   * attribute definition in HTML 4.0.
83   */
84  public String getMedia();
85  public void setMedia(String media);
86
87  /**
88   * Forward link type. See the rel attribute definition in HTML 4.0.
89   */
90  public String getRel();
91  public void setRel(String rel);
92
93  /**
94   * Reverse link type. See the rev attribute definition in HTML 4.0.
95   */
96  public String getRev();
97  public void setRev(String rev);
98
99  /**
100   * Frame to render the resource in. See the target attribute definition
101   * in HTML 4.0.
102   */
103  public String getTarget();
104  public void setTarget(String target);
```

```
105
106 /**
107  * Advisory content type. See the type attribute definition in HTML 4.0.
108  */
109 public String getType();
110 public void setType(String type);
111
112 }
```

### 40.33 HTMLMapElement.java

Secuencia 423: org/w3c/dom/html/HTMLMapElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
```

```

42 package org.w3c.dom.html;
43
44 /**
45  * Client-side image map. See the MAP element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
47  */
48 public interface HTMLMapElement extends HTMLElement {
49     /**
50      * The list of areas defined for the image map.
51      */
52     public HTMLCollection getAreas();
53
54     /**
55      * Names the map (for use with <code>usemap</code> ). See the name
56      * attribute definition in HTML 4.0.
57      */
58     public String getName();
59     public void setName(String name);
60
61 }

```

## 40.34 HTMLMenuElement.java

Secuencia 424: org/w3c/dom/html/HTMLMenuElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *

```

```

29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Menu list. See the MENU element definition in HTML 4.0. This element is
46  * deprecated in HTML 4.0.
47  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLMenuElement extends HTMLElement {
50     /**
51      * Reduce spacing between list items. See the compact attribute
52      * definition in HTML 4.0. This attribute is deprecated in HTML 4.0.
53      */
54     public boolean getCompact();
55     public void setCompact(boolean compact);
56
57 }

```

### 40.35 HTMLMetaElement.java

Secuencia 425: org/w3c/dom/html/HTMLMetaElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```



```
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * This contains generic meta-information about the document. See the META
46   * element definition in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48   * Model (DOM) Level 2 Specification</a>.
49   */
49  public interface HTMLMetaElement extends HTMLElement {
50      /**
51       * Associated information. See the content attribute definition in HTML
52       * 4.0.
53       */
54      public String getContent();
55      public void setContent(String content);
56
57      /**
58       * HTTP response header name. See the http-equiv attribute definition in
59       * HTML 4.0.
60       */
61      public String getHttpEquiv();
62      public void setHttpEquiv(String httpEquiv);
63
64      /**
65       * Meta information name. See the name attribute definition in HTML 4.0.
66       */
67      public String getName();
68      public void setName(String name);
69
70      /**
71       * Select form of content. See the scheme attribute definition in HTML
```

```
72     * 4.0.  
73     */  
74     public String getScheme();  
75     public void setScheme(String scheme);  
76  
77 }
```

#### 40.36 HTMLModElement.java

Secuencia 426: org/w3c/dom/html/HTMLModElement.java

```
1  /*  
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.  
3  *  
4  *  
5  *  
6  *  
7  *  
8  *  
9  *  
10 *  
11 *  
12 *  
13 *  
14 *  
15 *  
16 *  
17 *  
18 *  
19 *  
20 *  
21 *  
22 *  
23 */  
24  
25 /*  
26 *  
27 *  
28 *  
29 *  
30 *  
31 * Copyright (c) 2000 World Wide Web Consortium,  
32 * (Massachusetts Institute of Technology, Institut National de  
33 * Recherche en Informatique et en Automatique, Keio University). All  
34 * Rights Reserved. This program is distributed under the W3C's Software  
35 * Intellectual Property License. This program is distributed in the  
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even  
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR  
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more  
39 * details.  
40 */  
41  
42 package org.w3c.dom.html;  
43
```

```

44 /**
45  * Notice of modification to part of a document. See the  INS  and  DEL
46  * element definitions in HTML 4.0.
47  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLModElement extends HTMLElement {
50     /**
51      * A URI designating a document that describes the reason for the change.
52      * See the  cite  attribute definition in HTML 4.0.
53      */
54     public String getCite();
55     public void setCite(String cite);
56
57     /**
58      * The date and time of the change. See the  datetime  attribute definition
59      * in HTML 4.0.
60      */
61     public String getDateTime();
62     public void setDateTime(String dateTime);
63
64 }

```

## 40.37 HTMLListElement.java

Secuencia 427: org/w3c/dom/html/HTMLListElement.java

```

1  /**
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /**
26 *
27 *

```

```

28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Ordered list. See the OL element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  * Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLListElement extends HTMLElement {
50     /**
51      * Reduce spacing between list items. See the compact attribute
52      * definition in HTML 4.0. This attribute is deprecated in HTML 4.0.
53      */
54     public boolean getCompact();
55     public void setCompact(boolean compact);
56
57     /**
58      * Starting sequence number. See the start attribute definition in HTML
59      * 4.0. This attribute is deprecated in HTML 4.0.
60      */
61     public int getStart();
62     public void setStart(int start);
63
64     /**
65      * Numbering style. See the type attribute definition in HTML 4.0. This
66      * attribute is deprecated in HTML 4.0.
67      */
68     public String getType();
69     public void setType(String type);
70 }

```

## 40.38 HTMLObjectElement.java

Secuencia 428: org/w3c/dom/html/HTMLObjectElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *

```

```
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 import org.w3c.dom.Document;
45
46 /**
47  * Generic embedded object. Note. In principle, all properties on the object
48  * element are read-write but in some environments some properties may be
49  * read-only once the underlying object is instantiated. See the OBJECT
50  * element definition in HTML 4.0.
51  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
52  * Model (DOM) Level 2 Specification</a>.
53  */
54 public interface HTMLObjectElement extends HTMLElement {
55     /**
56      * Returns the <code>FORM</code> element containing this control. Returns
57      * <code>null</code> if this control is not within the context of a form.
58     */
59 }
```

```
58 public HTMLFormElement getForm();
59
60 /**
61  * Applet class file. See the <code>code</code> attribute for
62  * HTMLAppletElement.
63  */
64 public String getCode();
65 public void setCode(String code);
66
67 /**
68  * Aligns this object (vertically or horizontally) with respect to its
69  * surrounding text. See the align attribute definition in HTML 4.0.
70  * This attribute is deprecated in HTML 4.0.
71  */
72 public String getAlign();
73 public void setAlign(String align);
74
75 /**
76  * Space-separated list of archives. See the archive attribute definition
77  * in HTML 4.0.
78  */
79 public String getArchive();
80 public void setArchive(String archive);
81
82 /**
83  * Width of border around the object. See the border attribute definition
84  * in HTML 4.0. This attribute is deprecated in HTML 4.0.
85  */
86 public String getBorder();
87 public void setBorder(String border);
88
89 /**
90  * Base URI for <code>classid</code> , <code>data</code> , and
91  * <code>archive</code> attributes. See the codebase attribute definition
92  * in HTML 4.0.
93  */
94 public String getCodeBase();
95 public void setCodeBase(String codeBase);
96
97 /**
98  * Content type for data downloaded via <code>classid</code> attribute.
99  * See the codetype attribute definition in HTML 4.0.
100  */
101 public String getCodeType();
102 public void setCodeType(String codeType);
103
104 /**
105  * A URI specifying the location of the object's data. See the data
106  * attribute definition in HTML 4.0.
107  */
108 public String getData();
109 public void setData(String data);
110
```

```
111 /**
112  * Declare (for future reference), but do not instantiate, this object.
113  * See the declare attribute definition in HTML 4.0.
114  */
115 public boolean getDeclare();
116 public void setDeclare(boolean declare);
117
118 /**
119  * Override height. See the height attribute definition in HTML 4.0.
120  */
121 public String getHeight();
122 public void setHeight(String height);
123
124 /**
125  * Horizontal space to the left and right of this image, applet, or
126  * object. See the hspace attribute definition in HTML 4.0. This
127  * attribute is deprecated in HTML 4.0.
128  */
129 public String getHspace();
130 public void setHspace(String hspace);
131
132 /**
133  * Form control or object name when submitted with a form. See the name
134  * attribute definition in HTML 4.0.
135  */
136 public String getName();
137 public void setName(String name);
138
139 /**
140  * Message to render while loading the object. See the standby attribute
141  * definition in HTML 4.0.
142  */
143 public String getStandby();
144 public void setStandby(String standby);
145
146 /**
147  * Index that represents the element's position in the tabbing order. See
148  * the tabIndex attribute definition in HTML 4.0.
149  */
150 public int getTabIndex();
151 public void setTabIndex(int tabIndex);
152
153 /**
154  * Content type for data downloaded via <code>data</code> attribute. See
155  * the type attribute definition in HTML 4.0.
156  */
157 public String getType();
158 public void setType(String type);
159
160 /**
161  * Use client-side image map. See the usemap attribute definition in
162  * HTML 4.0.
163  */
```

```

164     public String getUseMap();
165     public void setUseMap(String useMap);
166
167     /**
168      * Vertical space above and below this image, applet, or object. See the
169      * vspace attribute definition in HTML 4.0. This attribute is deprecated
170      * in HTML 4.0.
171      */
172     public String getVspace();
173     public void setVspace(String vspace);
174
175     /**
176      * Override width. See the width attribute definition in HTML 4.0.
177      */
178     public String getWidth();
179     public void setWidth(String width);
180
181     /**
182      * The document this object contains, if there is any and it is
183      * available, or <code>null</code> otherwise.
184      * @since DOM Level 2
185      */
186     public Document getContentDocument();
187
188 }

```

#### 40.39 HTMLOptGroupElement.java

Secuencia 429: org/w3c/dom/html/HTMLOptGroupElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24

```



```

25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Group options together in logical subdivisions. See the OPTGROUP element
46   * definition in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
48   */
49  public interface HTMLOptGroupElement extends HTMLElement {
50      /**
51       * The control is unavailable in this context. See the disabled
52       * attribute definition in HTML 4.0.
53       */
54      public boolean getDisabled();
55      public void setDisabled(boolean disabled);
56
57      /**
58       * Assigns a label to this option group. See the label attribute
59       * definition in HTML 4.0.
60       */
61      public String getLabel();
62      public void setLabel(String label);
63
64  }

```

## 40.40 HTMLOptionElement.java

Secuencia 430: org/w3c/dom/html/HTMLOptionElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * A selectable choice. See the OPTION element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
47  */
48 public interface HTMLOptionElement extends HTMLElement {
49     /**
50      * Returns the <code>FORM</code> element containing this control. Returns
51      * <code>null</code> if this control is not within the context of a form.
52      */
53     public HTMLFormElement getForm();
54
55     /**
56      * Represents the value of the HTML selected attribute. The value of this
57      * attribute does not change if the state of the corresponding form
58      * control, in an interactive user agent, changes. Changing
59      * <code>defaultSelected</code> , however, resets the state of the form
60      * control. See the selected attribute definition in HTML 4.0.
```

```

61     */
62     public boolean getDefaultSelected();
63     public void setDefaultSelected(boolean defaultSelected);
64
65     /**
66      * The text contained within the option element.
67      */
68     public String getText();
69
70     /**
71      * The index of this <code>OPTION</code> in its parent <code>SELECT</code>
72      * , starting from 0.
73      */
74     public int getIndex();
75
76     /**
77      * The control is unavailable in this context. See the disabled
78      * attribute definition in HTML 4.0.
79      */
80     public boolean getDisabled();
81     public void setDisabled(boolean disabled);
82
83     /**
84      * Option label for use in hierarchical menus. See the label attribute
85      * definition in HTML 4.0.
86      */
87     public String getLabel();
88     public void setLabel(String label);
89
90     /**
91      * Represents the current state of the corresponding form control, in an
92      * interactive user agent. Changing this attribute changes the state of
93      * the form control, but does not change the value of the HTML selected
94      * attribute of the element.
95      */
96     public boolean getSelected();
97     public void setSelected(boolean selected);
98
99     /**
100      * The current form control value. See the value attribute definition in
101      * HTML 4.0.
102      */
103     public String getValue();
104     public void setValue(String value);
105
106 }

```

## 40.41 HTMLParagraphElement.java

Secuencia 431: org/w3c/dom/html/HTMLParagraphElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Paragraphs. See the P element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLParagraphElement extends HTMLElement {
50     /**
51      * Horizontal text alignment. See the align attribute definition in HTML
52      * 4.0. This attribute is deprecated in HTML 4.0.
53      */
54     public String getAlign();
55     public void setAlign(String align);
56 }
```

56 }

**40.42 HTMLParamElement.java**

Secuencia 432: org/w3c/dom/html/HTMLParamElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45 * Parameters fed to the <code>OBJECT</code> element. See the PARAM element
46 * definition in HTML 4.0.
47 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
   Model (DOM) Level 2 Specification</a>.
```

```

48  */
49  public interface HTMLParamElement extends HTMLElement {
50      /**
51       * The name of a run-time parameter. See the name attribute definition
52       * in HTML 4.0.
53       */
54      public String getName();
55      public void setName(String name);
56
57      /**
58       * Content type for the <code>value</code> attribute when
59       * <code>valuetype</code> has the value "ref". See the type attribute
60       * definition in HTML 4.0.
61       */
62      public String getType();
63      public void setType(String type);
64
65      /**
66       * The value of a run-time parameter. See the value attribute definition
67       * in HTML 4.0.
68       */
69      public String getValue();
70      public void setValue(String value);
71
72      /**
73       * Information about the meaning of the <code>value</code> attribute
74       * value. See the valuetype attribute definition in HTML 4.0.
75       */
76      public String getValueType();
77      public void setValueType(String valueType);
78
79  }

```

#### 40.43 HTMLPreElement.java

Secuencia 433: org/w3c/dom/html/HTMLPreElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *

```

```

18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Preformatted text. See the PRE element definition in HTML 4.0.
46   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
47   */
48  public interface HTMLPreElement extends HTMLElement {
49      /**
50       * Fixed width for content. See the width attribute definition in HTML
51       * 4.0. This attribute is deprecated in HTML 4.0.
52       */
53      public int getWidth();
54      public void setWidth(int width);
55
56  }

```

## 40.44 HTMLQuoteElement.java

Secuencia 434: org/w3c/dom/html/HTMLQuoteElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *

```

```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * For the <code>Q</code> and <code>BLOCKQUOTE</code> elements. See the Q
46   * element definition in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48   * Model (DOM) Level 2 Specification</a>.
49   */
49  public interface HTMLQuoteElement extends HTMLElement {
50      /**
51       * A URI designating a source document or message. See the cite
52       * attribute definition in HTML 4.0.
53       */
54      public String getCite();
55      public void setCite(String cite);
56
57  }

```

#### 40.45 HTMLScriptElement.java

Secuencia 435: org/w3c/dom/html/HTMLScriptElement.java



```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Script statements. See the SCRIPT element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLScriptElement extends HTMLElement {
50     /**
51      * The script content of the element.
52      */
53     public String getText();
54 }
```

```

53     public void setText(String text);
54
55     /**
56      * Reserved for future use.
57      */
58     public String getHtmlFor();
59     public void setHtmlFor(String htmlFor);
60
61     /**
62      * Reserved for future use.
63      */
64     public String getEvent();
65     public void setEvent(String event);
66
67     /**
68      * The character encoding of the linked resource. See the charset
69      * attribute definition in HTML 4.0.
70      */
71     public String getCharset();
72     public void setCharset(String charset);
73
74     /**
75      * Indicates that the user agent can defer processing of the script. See
76      * the defer attribute definition in HTML 4.0.
77      */
78     public boolean getDefer();
79     public void setDefer(boolean defer);
80
81     /**
82      * URI designating an external script. See the src attribute definition
83      * in HTML 4.0.
84      */
85     public String getSrc();
86     public void setSrc(String src);
87
88     /**
89      * The content type of the script language. See the type attribute
90      * definition in HTML 4.0.
91      */
92     public String getType();
93     public void setType(String type);
94
95 }

```

#### 40.46 HTMLSelectElement.java

Secuencia 436: org/w3c/dom/html/HTMLSelectElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 import org.w3c.dom.DOMException;
45
46 /**
47  * The select element allows the selection of an option. The contained
48  * options can be directly accessed through the select element as a
49  * collection. See the SELECT element definition in HTML 4.0.
50  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
51  */
52 public interface HTMLSelectElement extends HTMLElement {
53     /**
54      * The type of this form control. This is the string "select-multiple"
55      * when the multiple attribute is <code>true</code> and the string
56      * "select-one" when <code>false</code> .
57      */
58     public String getType();
```

```
59
60 /**
61  * The ordinal index of the selected option, starting from 0. The value
62  * -1 is returned if no element is selected. If multiple options are
63  * selected, the index of the first selected option is returned.
64  */
65 public int getSelectedIndex();
66 public void setSelectedIndex(int selectedIndex);
67
68 /**
69  * The current form control value.
70  */
71 public String getValue();
72 public void setValue(String value);
73
74 /**
75  * The number of options in this SELECT .
76  */
77 public int getLength();
78
79 /**
80  * Returns the FORM element containing this control. Returns
81  * null if this control is not within the context of a form.
82  */
83 public HTMLFormElement getForm();
84
85 /**
86  * The collection of OPTION elements contained by this
87  * element.
88  */
89 public HTMLCollection getOptions();
90
91 /**
92  * The control is unavailable in this context. See the disabled
93  * attribute definition in HTML 4.0.
94  */
95 public boolean getDisabled();
96 public void setDisabled(boolean disabled);
97
98 /**
99  * If true, multiple OPTION elements may be selected in
100  * this SELECT . See the multiple attribute definition in
101  * HTML 4.0.
102  */
103 public boolean getMultiple();
104 public void setMultiple(boolean multiple);
105
106 /**
107  * Form control or object name when submitted with a form. See the name
108  * attribute definition in HTML 4.0.
109  */
110 public String getName();
111 public void setName(String name);
```

```
112
113 /**
114  * Number of visible rows. See the size attribute definition in HTML 4.0.
115  */
116 public int getSize();
117 public void setSize(int size);
118
119 /**
120  * Index that represents the element's position in the tabbing order. See
121  * the tabIndex attribute definition in HTML 4.0.
122  */
123 public int getTabIndex();
124 public void setTabIndex(int tabIndex);
125
126 /**
127  * Add a new element to the collection of <code>OPTION</code> elements
128  * for this <code>SELECT</code> . This method is the equivalent of the
129  * <code>appendChild</code> method of the <code>Node</code> interface if
130  * the <code>before</code> parameter is <code>null</code> . It is
131  * equivalent to the <code>insertBefore</code> method on the parent of
132  * <code>before</code> in all other cases.
133  * @param element The element to add.
134  * @param before The element to insert before, or <code>null</code> for
135  * the tail of the list.
136  * @exception DOMException
137  * NOT_FOUND_ERR: Raised if <code>before</code> is not a descendant of
138  * the <code>SELECT</code> element.
139  */
140 public void add(HTMLElement element,
141                HTMLElement before)
142                throws DOMException;
143
144 /**
145  * Remove an element from the collection of <code>OPTION</code> elements
146  * for this <code>SELECT</code> . Does nothing if no element has the given
147  * index.
148  * @param index The index of the item to remove, starting from 0.
149  */
150 public void remove(int index);
151
152 /**
153  * Removes keyboard focus from this element.
154  */
155 public void blur();
156
157 /**
158  * Gives keyboard focus to this element.
159  */
160 public void focus();
161
162 }
```

## 40.47 HTMLStyleElement.java

Secuencia 437: org/w3c/dom/html/HTMLStyleElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45 * Style information. See the STYLE element definition in HTML 4.0, the
46 * module and the <code>LinkStyle</code> interface in the module.
47 * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48 * Model (DOM) Level 2 Specification</a>.
49 */
49 public interface HTMLStyleElement extends HTMLElement {
```

```
50  /**
51   * Enables/disables the style sheet.
52   */
53  public boolean getDisabled();
54  public void setDisabled(boolean disabled);
55
56  /**
57   * Designed for use with one or more target media. See the media
58   * attribute definition in HTML 4.0.
59   */
60  public String getMedia();
61  public void setMedia(String media);
62
63  /**
64   * The content type pf the style sheet language. See the type attribute
65   * definition in HTML 4.0.
66   */
67  public String getType();
68  public void setType(String type);
69
70 }
```

#### 40.48 HTMLTableCaptionElement.java

Secuencia 438: org/w3c/dom/html/HTMLTableCaptionElement.java

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
```

```

29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Table caption See the CAPTION element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLTableCaptionElement extends HTMLElement {
50     /**
51      * Caption alignment with respect to the table. See the align attribute
52      * definition in HTML 4.0. This attribute is deprecated in HTML 4.0.
53      */
54     public String getAlign();
55     public void setAlign(String align);
56 }

```

#### 40.49 HTMLTableCellElement.java

Secuencia 439: org/w3c/dom/html/HTMLTableCellElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```



```
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * The object used to represent the <code>TH</code> and <code>TD</code>
46   * elements. See the TD element definition in HTML 4.0.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
48   * Model (DOM) Level 2 Specification</a>.
49   */
49  public interface HTMLTableCellElement extends HTMLElement {
50      /**
51       * The index of this cell in the row, starting from 0. This index is in
52       * document tree order and not display order.
53       */
54      public int getCellIndex();
55
56      /**
57       * Abbreviation for header cells. See the abbr attribute definition in
58       * HTML 4.0.
59       */
60      public String getAbbr();
61      public void setAbbr(String abbr);
62
63      /**
64       * Horizontal alignment of data in cell. See the align attribute
65       * definition in HTML 4.0.
66       */
67      public String getAlign();
68      public void setAlign(String align);
69
70      /**
71       * Names group of related headers. See the axis attribute definition in
72       * HTML 4.0.
```

```
73     */
74     public String getAxis();
75     public void setAxis(String axis);
76
77     /**
78      * Cell background color. See the bgcolor attribute definition in HTML
79      * 4.0. This attribute is deprecated in HTML 4.0.
80      */
81     public String getBgColor();
82     public void setBgColor(String bgColor);
83
84     /**
85      * Alignment character for cells in a column. See the char attribute
86      * definition in HTML 4.0.
87      */
88     public String getCh();
89     public void setCh(String ch);
90
91     /**
92      * Offset of alignment character. See the charoff attribute definition
93      * in HTML 4.0.
94      */
95     public String getChOff();
96     public void setChOff(String chOff);
97
98     /**
99      * Number of columns spanned by cell. See the colspan attribute
100     * definition in HTML 4.0.
101     */
102     public int getColSpan();
103     public void setColSpan(int colSpan);
104
105     /**
106      * List of <code>id</code> attribute values for header cells. See the
107      * headers attribute definition in HTML 4.0.
108      */
109     public String getHeaders();
110     public void setHeaders(String headers);
111
112     /**
113      * Cell height. See the height attribute definition in HTML 4.0. This
114      * attribute is deprecated in HTML 4.0.
115      */
116     public String getHeight();
117     public void setHeight(String height);
118
119     /**
120      * Suppress word wrapping. See the nowrap attribute definition in HTML
121      * 4.0. This attribute is deprecated in HTML 4.0.
122      */
123     public boolean getNoWrap();
124     public void setNoWrap(boolean noWrap);
125
```

```
126  /**
127   * Number of rows spanned by cell. See the rowspan attribute definition
128   * in HTML 4.0.
129   */
130  public int getRowSpan();
131  public void setRowSpan(int rowSpan);
132
133  /**
134   * Scope covered by header cells. See the scope attribute definition in
135   * HTML 4.0.
136   */
137  public String getScope();
138  public void setScope(String scope);
139
140  /**
141   * Vertical alignment of data in cell. See the valign attribute
142   * definition in HTML 4.0.
143   */
144  public String getVAlign();
145  public void setVAlign(String vAlign);
146
147  /**
148   * Cell width. See the width attribute definition in HTML 4.0. This
149   * attribute is deprecated in HTML 4.0.
150   */
151  public String getWidth();
152  public void setWidth(String width);
153
154 }
```

#### 40.50 HTMLTableColElement.java

Secuencia 440: [org.w3c/dom/html/HTMLTableColElement.java](http://org.w3c/dom/html/HTMLTableColElement.java)

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
```

```

21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Regroups the COL and COLGROUP elements. See the
46   * COL element definition in HTML 4.0.
47   * 

See also the http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510 Document Object
48   * Model (DOM) Level 2 Specification


49   */
49  public interface HTMLTableColElement extends HTMLElement {
50      /**
51       * Horizontal alignment of cell data in column. See the align attribute
52       * definition in HTML 4.0.
53       */
54      public String getAlign();
55      public void setAlign(String align);
56
57      /**
58       * Alignment character for cells in a column. See the char attribute
59       * definition in HTML 4.0.
60       */
61      public String getCh();
62      public void setCh(String ch);
63
64      /**
65       * Offset of alignment character. See the charoff attribute definition
66       * in HTML 4.0.
67       */
68      public String getChOff();
69      public void setChOff(String chOff);
70
71      /**
72       * Indicates the number of columns in a group or affected by a grouping.

```

```
73     * See the span attribute definition in HTML 4.0.
74     */
75     public int getSpan();
76     public void setSpan(int span);
77
78     /**
79     * Vertical alignment of cell data in column. See the valign attribute
80     * definition in HTML 4.0.
81     */
82     public String getVAlign();
83     public void setVAlign(String vAlign);
84
85     /**
86     * Default column width. See the width attribute definition in HTML 4.0.
87     */
88     public String getWidth();
89     public void setWidth(String width);
90
91 }
```

## 40.51 HTMLTableElement.java

Secuencia 441: org/w3c/dom/html/HTMLTableElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
```

```
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 import org.w3c.dom.DOMException;
45
46 /**
47  * The create* and delete* methods on the table allow authors to construct
48  * and modify tables. HTML 4.0 specifies that only one of each of the
49  * <code>CAPTION</code> , <code>THEAD</code> , and <code>TFOOT</code>
50  * elements may exist in a table. Therefore, if one exists, and the
51  * createTHead() or createTFoot() method is called, the method returns the
52  * existing THead or TFoot element. See the TABLE element definition in HTML
53  * 4.0.
54  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
55  * Model (DOM) Level 2 Specification</a>.
56  */
57 public interface HTMLTableElement extends HTMLElement {
58     /**
59      * Returns the table's <code>CAPTION</code> , or void if none exists.
60      */
61     public HTMLTableCaptionElement getCaption();
62     public void setCaption(HTMLTableCaptionElement caption);
63
64     /**
65      * Returns the table's <code>THEAD</code> , or <code>null</code> if none
66      * exists.
67      */
68     public HTMLTableSectionElement getTHead();
69     public void setTHead(HTMLTableSectionElement tHead);
70
71     /**
72      * Returns the table's <code>TFOOT</code> , or <code>null</code> if none
73      * exists.
74      */
75     public HTMLTableSectionElement getTFoot();
76     public void setTFoot(HTMLTableSectionElement tFoot);
77
78     /**
79      * Returns a collection of all the rows in the table, including all in
80      * <code>THEAD</code> , <code>TFOOT</code> , all <code>TBODY</code>
81      * elements.
82      */
83     public HTMLCollection getRows();
```

```
83
84 /**
85  * Returns a collection of the defined table bodies.
86  */
87 public HTMLCollection getTBodies();
88
89 /**
90  * Specifies the table's position with respect to the rest of the
91  * document. See the align attribute definition in HTML 4.0. This
92  * attribute is deprecated in HTML 4.0.
93  */
94 public String getAlign();
95 public void setAlign(String align);
96
97 /**
98  * Cell background color. See the bgcolor attribute definition in HTML
99  * 4.0. This attribute is deprecated in HTML 4.0.
100  */
101 public String getBgColor();
102 public void setBgColor(String bgColor);
103
104 /**
105  * The width of the border around the table. See the border attribute
106  * definition in HTML 4.0.
107  */
108 public String getBorder();
109 public void setBorder(String border);
110
111 /**
112  * Specifies the horizontal and vertical space between cell content and
113  * cell borders. See the cellpadding attribute definition in HTML 4.0.
114  */
115 public String getCellPadding();
116 public void setCellPadding(String cellPadding);
117
118 /**
119  * Specifies the horizontal and vertical separation between cells. See
120  * the cellspacing attribute definition in HTML 4.0.
121  */
122 public String getCellSpacing();
123 public void setCellSpacing(String cellSpacing);
124
125 /**
126  * Specifies which external table borders to render. See the frame
127  * attribute definition in HTML 4.0.
128  */
129 public String getFrame();
130 public void setFrame(String frame);
131
132 /**
133  * Specifies which internal table borders to render. See the rules
134  * attribute definition in HTML 4.0.
135  */
```

```
136 public String getRules();
137 public void setRules(String rules);
138
139 /**
140  * Description about the purpose or structure of a table. See the
141  * summary attribute definition in HTML 4.0.
142  */
143 public String getSummary();
144 public void setSummary(String summary);
145
146 /**
147  * Specifies the desired table width. See the width attribute definition
148  * in HTML 4.0.
149  */
150 public String getWidth();
151 public void setWidth(String width);
152
153 /**
154  * Create a table header row or return an existing one.
155  * @return A new table header element (<code>THEAD</code> ).
156  */
157 public HTMLElement createTHead();
158
159 /**
160  * Delete the header from the table, if one exists.
161  */
162 public void deleteTHead();
163
164 /**
165  * Create a table footer row or return an existing one.
166  * @return A footer element (<code>TFOOT</code> ).
167  */
168 public HTMLElement createTFoot();
169
170 /**
171  * Delete the footer from the table, if one exists.
172  */
173 public void deleteTFoot();
174
175 /**
176  * Create a new table caption object or return an existing one.
177  * @return A <code>CAPTION</code> element.
178  */
179 public HTMLElement createCaption();
180
181 /**
182  * Delete the table caption, if one exists.
183  */
184 public void deleteCaption();
185
186 /**
187  * Insert a new empty row in the table. The new row is inserted
188  * immediately before and in the same section as the current
```



```

189  * <code>index</code> th row in the table. If <code>index</code> is equal
190  * to the number of rows, the new row is appended. In addition, when the
191  * table is empty the row is inserted into a <code>TBODY</code> which is
192  * created and inserted into the table. Note. A table row cannot be empty
193  * according to HTML 4.0 Recommendation.
194  * @param index The row number where to insert a new row. This index
195  * starts from 0 and is relative to all the rows contained inside the
196  * table, regardless of section parentage.
197  * @return The newly created row.
198  * @exception DOMException
199  *     INDEX_SIZE_ERR: Raised if the specified index is greater than the
200  *     number of rows or if the index is negative.
201  */
202  public HTMLElement insertRow(int index)
203      throws DOMException;
204
205  /**
206   * Delete a table row.
207   * @param index The index of the row to be deleted. This index starts
208   * from 0 and is relative to all the rows contained inside the table,
209   * regardless of section parentage.
210   * @exception DOMException
211   *     INDEX_SIZE_ERR: Raised if the specified index is greater than or
212   *     equal to the number of rows or if the index is negative.
213   */
214  public void deleteRow(int index)
215      throws DOMException;
216
217  }

```

## 40.52 HTMLTableRowElement.java

Secuencia 442: org/w3c/dom/html/HTMLTableRowElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *

```

```

21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  import org.w3c.dom.DOMException;
45
46  /**
47   * A row in a table. See the TR element definition in HTML 4.0.
48   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
49   */
50  public interface HTMLTableRowElement extends HTMLElement {
51      /**
52       * The index of this row, relative to the entire table, starting from 0.
53       * This is in document tree order and not display order. The
54       * <code>rowIndex</code> does not take into account sections (
55       * <code>THEAD</code> , <code>TFOOT</code> , or <code>TBODY</code> )
56       * within the table.
57       */
58      public int getRowIndex();
59
60      /**
61       * The index of this row, relative to the current section (
62       * <code>THEAD</code> , <code>TFOOT</code> , or <code>TBODY</code> ),
63       * starting from 0.
64       */
65      public int getSectionRowIndex();
66
67      /**
68       * The collection of cells in this row.
69       */
70      public HTMLCollection getCells();
71
72      /**

```

```

73     * Horizontal alignment of data within cells of this row. See the align
74     * attribute definition in HTML 4.0.
75     */
76     public String getAlign();
77     public void setAlign(String align);
78
79     /**
80     * Background color for rows. See the bgcolor attribute definition in
81     * HTML 4.0. This attribute is deprecated in HTML 4.0.
82     */
83     public String getBgColor();
84     public void setBgColor(String bgColor);
85
86     /**
87     * Alignment character for cells in a column. See the char attribute
88     * definition in HTML 4.0.
89     */
90     public String getCh();
91     public void setCh(String ch);
92
93     /**
94     * Offset of alignment character. See the charoff attribute definition
95     * in HTML 4.0.
96     */
97     public String getChOff();
98     public void setChOff(String chOff);
99
100    /**
101    * Vertical alignment of data within cells of this row. See the valign
102    * attribute definition in HTML 4.0.
103    */
104    public String getVAlign();
105    public void setVAlign(String vAlign);
106
107    /**
108    * Insert an empty <code>TD</code> cell into this row. If
109    * <code>index</code> is equal to the number of cells, the new cell is
110    * appended
111    * @param index The place to insert the cell, starting from 0.
112    * @return The newly created cell.
113    * @exception DOMException
114    *     INDEX_SIZE_ERR: Raised if the specified <code>index</code> is
115    *     greater than the number of cells or if the index is negative.
116    */
117    public HTMLElement insertCell(int index)
118        throws DOMException;
119
120    /**
121    * Delete a cell from the current row.
122    * @param index The index of the cell to delete, starting from 0.
123    * @exception DOMException
124    *     INDEX_SIZE_ERR: Raised if the specified <code>index</code> is
125    *     greater than or equal to the number of cells or if the index is

```

```
126     *   negative.
127     */
128     public void deleteCell(int index)
129         throws DOMException;
130
131 }
```

#### 40.53 HTMLTableSectionElement.java

Secuencia 443: org/w3c/dom/html/HTMLTableSectionElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
```

```

44 import org.w3c.dom.DOMException;
45
46 /**
47  * The <code>THEAD</code> , <code>TFOOT</code> , and <code>TBODY</code>
48  * elements.
49  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
50  */
51 public interface HTMLTableSectionElement extends HTMLElement {
52     /**
53      * Horizontal alignment of data in cells. See the <code>align</code>
54      * attribute for HTMLTheadElement for details.
55      */
56     public String getAlign();
57     public void setAlign(String align);
58
59     /**
60      * Alignment character for cells in a column. See the char attribute
61      * definition in HTML 4.0.
62      */
63     public String getCh();
64     public void setCh(String ch);
65
66     /**
67      * Offset of alignment character. See the charoff attribute definition
68      * in HTML 4.0.
69      */
70     public String getChOff();
71     public void setChOff(String chOff);
72
73     /**
74      * Vertical alignment of data in cells. See the <code>valign</code>
75      * attribute for HTMLTheadElement for details.
76      */
77     public String getVAlign();
78     public void setVAlign(String vAlign);
79
80     /**
81      * The collection of rows in this table section.
82      */
83     public HTMLCollection getRows();
84
85     /**
86      * Insert a row into this section. The new row is inserted immediately
87      * before the current <code>index</code> th row in this section. If
88      * <code>index</code> is equal to the number of rows in this section, the
89      * new row is appended.
90      * @param index The row number where to insert a new row. This index
91      * starts from 0 and is relative only to the rows contained inside this
92      * section, not all the rows in the table.
93      * @return The newly created row.
94      * @exception DOMException
95      * INDEX_SIZE_ERR: Raised if the specified index is greater than the

```

```

96     *   number of rows of if the index is neagative.
97     */
98     public HTMLElement insertRow(int index)
99         throws DOMException;
100
101     /**
102     *   Delete a row from this section.
103     *   @param index   The index of the row to be deleted. This index starts
104     *   from 0 and is relative only to the rows contained inside this
105     *   section, not all the rows in the table.
106     *   @exception DOMException
107     *   INDEX_SIZE_ERR: Raised if the specified index is greater than or
108     *   equal to the number of rows or if the index is negative.
109     */
110     public void deleteRow(int index)
111         throws DOMException;
112
113 }

```

#### 40.54 HTMLTextAreaElement.java

Secuencia 444: org/w3c/dom/html/HTMLTextAreaElement.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,

```

```
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39 * details.
40 */
41
42 package org.w3c.dom.html;
43
44 /**
45  * Multi-line text field. See the TEXTAREA element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
47  *   Model (DOM) Level 2 Specification</a>.
48  */
49 public interface HTMLTextAreaElement extends HTMLElement {
50     /**
51      * Represents the contents of the element. The value of this attribute
52      * does not change if the contents of the corresponding form control, in
53      * an interactive user agent, changes. Changing this attribute, however,
54      * resets the contents of the form control.
55      */
56     public String getDefaultValue();
57     public void setDefaultValue(String defaultValue);
58
59     /**
60      * Returns the <code>FORM</code> element containing this control. Returns
61      * <code>null</code> if this control is not within the context of a form.
62      */
63     public HTMLFormElement getForm();
64
65     /**
66      * A single character access key to give access to the form control. See
67      * the accesskey attribute definition in HTML 4.0.
68      */
69     public String getAccessKey();
70     public void setAccessKey(String accessKey);
71
72     /**
73      * Width of control (in characters). See the cols attribute definition
74      * in HTML 4.0.
75      */
76     public int getCols();
77     public void setCols(int cols);
78
79     /**
80      * The control is unavailable in this context. See the disabled
81      * attribute definition in HTML 4.0.
82      */
83     public boolean getDisabled();
84     public void setDisabled(boolean disabled);
```

```
84
85 /**
86  * Form control or object name when submitted with a form. See the name
87  * attribute definition in HTML 4.0.
88  */
89 public String getName();
90 public void setName(String name);
91
92 /**
93  * This control is read-only. See the readonly attribute definition in
94  * HTML 4.0.
95  */
96 public boolean getReadOnly();
97 public void setReadOnly(boolean readOnly);
98
99 /**
100  * Number of text rows. See the rows attribute definition in HTML 4.0.
101  */
102 public int getRows();
103 public void setRows(int rows);
104
105 /**
106  * Index that represents the element's position in the tabbing order. See
107  * the tabIndex attribute definition in HTML 4.0.
108  */
109 public int getTabIndex();
110 public void setTabIndex(int tabIndex);
111
112 /**
113  * The type of this form control. This the string "textarea".
114  */
115 public String getType();
116
117 /**
118  * Represents the current contents of the corresponding form control, in
119  * an interactive user agent. Changing this attribute changes the
120  * contents of the form control, but does not change the contents of the
121  * element. If the entirety of the data can not fit into a single
122  * <code>DOMString</code> , the implementation may truncate the data.
123  */
124 public String getValue();
125 public void setValue(String value);
126
127 /**
128  * Removes keyboard focus from this element.
129  */
130 public void blur();
131
132 /**
133  * Gives keyboard focus to this element.
134  */
135 public void focus();
136
```



```
137  /**
138   * Select the contents of the <code>TEXTAREA</code> .
139   */
140   public void select();
141
142 }
```

## 40.55 HTMLTitleElement.java

Secuencia 445: org/w3c/dom/html/HTMLTitleElement.java

```
1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26   *
27   *
28   *
29   *
30   *
31   * Copyright (c) 2000 World Wide Web Consortium,
32   * (Massachusetts Institute of Technology, Institut National de
33   * Recherche en Informatique et en Automatique, Keio University). All
34   * Rights Reserved. This program is distributed under the W3C's Software
35   * Intellectual Property License. This program is distributed in the
36   * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37   * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38   * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39   * details.
40   */
41
42  package org.w3c.dom.html;
43
```

```
44 /**
45  * The document title. See the TITLE element definition in HTML 4.0.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
    Model (DOM) Level 2 Specification</a>.
47  */
48 public interface HTMLTitleElement extends HTMLElement {
49     /**
50      * The specified title as a string.
51      */
52     public String getText();
53     public void setText(String text);
54 }
55 }
```

## 40.56 HTMLUListElement.java

Secuencia 446: org/w3c/dom/html/HTMLUListElement.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
```

```

37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE. See W3C License http://www.w3.org/Consortium/Legal/ for more
39  * details.
40  */
41
42  package org.w3c.dom.html;
43
44  /**
45   * Unordered list. See the UL element definition in HTML 4.0.
46   * <p>See also the <a href='http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510'>Document Object
      Model (DOM) Level 2 Specification</a>.
47   */
48  public interface HTMLULListElement extends HTMLElement {
49      /**
50       * Reduce spacing between list items. See the compact attribute
51       * definition in HTML 4.0. This attribute is deprecated in HTML 4.0.
52       */
53      public boolean getCompact();
54      public void setCompact(boolean compact);
55
56      /**
57       * Bullet style. See the type attribute definition in HTML 4.0. This
58       * attribute is deprecated in HTML 4.0.
59       */
60      public String getType();
61      public void setType(String type);
62
63  }

```

## 41 Dom Ls (org.w3c.dom.ls package)

### 41.1 DOMImplementationLS.java

Secuencia 447: org/w3c/dom/ls/DOMImplementationLS.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```

```

20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom.ls;
43
44  import org.w3c.dom.DOMException;
45
46  /**
47   * <code>DOMImplementationLS</code> contains the factory methods for creating
48   * Load and Save objects.
49   * <p> The expectation is that an instance of the
50   * <code>DOMImplementationLS</code> interface can be obtained by using
51   * binding-specific casting methods on an instance of the
52   * <code>DOMImplementation</code> interface or, if the <code>Document</code>
53   * supports the feature <code>"Core"</code> version <code>"3.0"</code>
54   * defined in [

```

```

71
72 /**
73  * Create a new LSParser. The newly constructed parser may
74  * then be configured by means of its DOMConfiguration
75  * object, and used to parse documents by means of its parse
76  * method.
77  * @param mode The mode argument is either
78  * MODE_SYNCHRONOUS or MODE_ASYNCHRONOUS, if
79  * mode is MODE_SYNCHRONOUS then the
80  * LSParser that is created will operate in synchronous
81  * mode, if it's MODE_ASYNCHRONOUS then the
82  * LSParser that is created will operate in asynchronous
83  * mode.
84  * @param schemaType An absolute URI representing the type of the schema
85  * language used during the load of a Document using the
86  * newly created LSParser. Note that no lexical checking
87  * is done on the absolute URI. In order to create a
88  * LSParser for any kind of schema types (i.e. the
89  * LSParser will be free to use any schema found), use the value
90  * null.
91  * 

><b>Note:</b> For W3C XML Schema ["http://www.w3.org/2001/XMLSchema". For XML DTD \["http://www.w3.org/TR/REC-xml". Other Schema languages
96  \\* are outside the scope of the W3C and therefore should recommend an
97  \\* absolute URI in order to use this method.
98  \\* @return The newly created LSParser object. This
99  \\* LSParser is either synchronous or asynchronous
100  \\* depending on the value of the mode argument.
101  \\* 

><b>Note:</b> By default, the newly created LSParser
102  \\* does not contain a DOMErrorHandler, i.e. the value of
103  \\* the "null. However, implementations
105  \\\* may provide a default error handler at creation time. In that case,
106  \\\* the initial value of the "error-handler" configuration
107  \\\* parameter on the new LSParser object contains a
108  \\\* reference to the default error handler.
109  \\\* @exception DOMException
110  \\\* NOT\\\_SUPPORTED\\\_ERR: Raised if the requested mode or schema type is
111  \\\* not supported.
112  \\\*/
113 public LSParser createLSParser\\\(short mode,
114                               String schemaType\\\)
115     throws DOMException;
116
117 /\\\*\\\*
118  \\\* Create a new LSSerializer object.
119  \\\* @return The newly created LSSerializer object.
120  \\\* 

><b>Note:</b> By default, the newly created
121  \\\* LSSerializer has no DOMErrorHandler, i.e.


```

```

122     * the value of the <code>"error-handler"</code> configuration
123     * parameter is <code>null</code>. However, implementations may
124     * provide a default error handler at creation time. In that case, the
125     * initial value of the <code>"error-handler"</code> configuration
126     * parameter on the new <code>LSSerializer</code> object contains a
127     * reference to the default error handler.
128     */
129     public LSSerializer createLSSerializer();
130
131     /**
132     * Create a new empty input source object where
133     * <code>LSInput.characterStream</code>, <code>LSInput.byteStream</code>
134     * , <code>LSInput.stringData</code> <code>LSInput.systemId</code>,
135     * <code>LSInput.publicId</code>, <code>LSInput.baseURI</code>, and
136     * <code>LSInput.encoding</code> are null, and
137     * <code>LSInput.certifiedText</code> is false.
138     * @return The newly created input object.
139     */
140     public LSInput createLSInput();
141
142     /**
143     * Create a new empty output destination object where
144     * <code>LSOutput.characterStream</code>,
145     * <code>LSOutput.byteStream</code>, <code>LSOutput.systemId</code>,
146     * <code>LSOutput.encoding</code> are null.
147     * @return The newly created output object.
148     */
149     public LSOutput createLSOutput();
150
151 }

```

## 41.2 LSEException.java

Secuencia 448: org/w3c/dom/ls/LSEException.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *

```

```

20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom.ls;
43
44  /**
45   * Parser or write operations may throw an LSEException if the
46   * processing is stopped. The processing can be stopped due to a
47   * DOMError with a severity of
48   * DOMError.SEVERITY_FATAL_ERROR or a non recovered
49   * DOMError.SEVERITY_ERROR, or if
50   * DOMErrorHandler.handleError() returned false.
51   * 

><b>Note:</b> As suggested in the definition of the constants in the
52   * DOMError interface, a DOM implementation may choose to
53   * continue after a fatal error, but the resulting DOM tree is then
54   * implementation dependent.
55   * 

>See also the Document Object
56   \* Model \(DOM\) Level 3 Load
57   \* and Save Specification.
58   */
59  public class LSEException extends RuntimeException {
60      public LSEException(short code, String message) {
61          super(message);
62          this.code = code;
63      }
64      public short code;
65      // LSEExceptionCode
66      /**
67       * If an attempt was made to load a document, or an XML Fragment, using
68       * LSParser and the processing has been stopped.
69       */
70      public static final short PARSE_ERR = 81;
71      /**
72       * If an attempt was made to serialize a Node using


```

```
72     * <code>LSSerializer</code> and the processing has been stopped.
73     */
74     public static final short SERIALIZE_ERR          = 82;
75
76 }
```

### 41.3 LSInput.java

Secuencia 449: org/w3c/dom/ls/LSInput.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom.ls;
43
44 /**
```



```

45  * This interface represents an input source for data.
46  * <p> This interface allows an application to encapsulate information about
47  * an input source in a single object, which may include a public
48  * identifier, a system identifier, a byte stream (possibly with a specified
49  * encoding), a base URI, and/or a character stream.
50  * <p> The exact definitions of a byte stream and a character stream are
51  * binding dependent.
52  * <p> The application is expected to provide objects that implement this
53  * interface whenever such objects are needed. The application can either
54  * provide its own objects that implement this interface, or it can use the
55  * generic factory method DOMImplementationLS.createLSInput()
56  * to create objects that implement this interface.
57  * <p> The LSParser will use the LSInput object to
58  * determine how to read data. The LSParser will look at the
59  * different inputs specified in the LSInput in the following
60  * order to know which one to read from, the first one that is not null and
61  * not an empty string will be used:
62  * <ol>
63  * <li> LSInput.characterStream
64  * </li>
65  * <li>
66  * <code>LSInput.byteStream</code>
67  * </li>
68  * <li> LSInput.stringData
69  * </li>
70  * <li>
71  * <code>LSInput.systemId</code>
72  * </li>
73  * <li> LSInput.publicId
74  * </li>
75  * </ol>
76  * <p> If all inputs are null, the LSParser will report a
77  * DOMError with its DOMError.type set to
78  * "no-input-specified" and its DOMError.severity
79  * set to DOMError.SEVERITY_FATAL_ERROR.
80  * <p> LSInput objects belong to the application. The DOM
81  * implementation will never modify them (though it may make copies and
82  * modify the copies, if necessary).
83  * <p> See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407'>Document Object
      Model (DOM) Level 3 Load
84  and Save Specification</a>.
85  */
86  public interface LSInput {
87      /**
88       * An attribute of a language and binding dependent type that represents
89       * a stream of 16-bit units. The application must encode the stream
90       * using UTF-16 (defined in [Unicode] and in [ISO/IEC 10646]). It is not a requirement to have
          an XML declaration when
91       * using character streams. If an XML declaration is present, the value
92       * of the encoding attribute will be ignored.
93       */
94       public java.io.Reader getCharacterStream();
95       /**

```

```
96      * An attribute of a language and binding dependent type that represents
97      * a stream of 16-bit units. The application must encode the stream
98      * using UTF-16 (defined in [Unicode] and in [ISO/IEC 10646]). It is not a requirement to have
    an XML declaration when
99      * using character streams. If an XML declaration is present, the value
100     * of the encoding attribute will be ignored.
101     */
102     public void setCharacterStream(java.io.Reader characterStream);
103
104     /**
105     * An attribute of a language and binding dependent type that represents
106     * a stream of bytes.
107     * <br> If the application knows the character encoding of the byte
108     * stream, it should set the encoding attribute. Setting the encoding in
109     * this way will override any encoding specified in an XML declaration
110     * in the data.
111     */
112     public java.io.InputStream getByteStream();
113
114     /**
115     * An attribute of a language and binding dependent type that represents
116     * a stream of bytes.
117     * <br> If the application knows the character encoding of the byte
118     * stream, it should set the encoding attribute. Setting the encoding in
119     * this way will override any encoding specified in an XML declaration
120     * in the data.
121     */
122     public void setByteStream(java.io.InputStream byteStream);
123
124     /**
125     * String data to parse. If provided, this will always be treated as a
126     * sequence of 16-bit units (UTF-16 encoded characters). It is not a
127     * requirement to have an XML declaration when using
128     * <code>stringData</code>. If an XML declaration is present, the value
129     * of the encoding attribute will be ignored.
130     */
131     public String getStringData();
132
133     /**
134     * String data to parse. If provided, this will always be treated as a
135     * sequence of 16-bit units (UTF-16 encoded characters). It is not a
136     * requirement to have an XML declaration when using
137     * <code>stringData</code>. If an XML declaration is present, the value
138     * of the encoding attribute will be ignored.
139     */
140     public void setStringData(String stringData);
141
142     /**
143     * The system identifier, a URI reference [<a href='http://www.ietf.org/rfc/rfc2396.txt'>IETF
    RFC 2396</a>], for this
144     * input source. The system identifier is optional if there is a byte
145     * stream, a character stream, or string data. It is still useful to
146     * provide one, since the application will use it to resolve any
    relative URIs and can include it in error messages and warnings. (The
    LSParser will only attempt to fetch the resource identified by the
```

```

147 * URI reference if there is no other input available in the input
148 * source.)
149 * <br> If the application knows the character encoding of the object
150 * pointed to by the system identifier, it can set the encoding using
151 * the <code>encoding</code> attribute.
152 * <br> If the specified system ID is a relative URI reference (see
153 * section 5 in [

```

```

198     * resolving a relative <code>systemId</code> to an absolute URI.
199     * <br> If, when used, the base URI is itself a relative URI, an empty
200     * string, or null, the behavior is implementation dependent.
201     */
202     public String getBaseURI();
203     /**
204     * The base URI to be used (see section 5.1.4 in [

```

```
246
247 }
```

#### 41.4 LSEvent.java

Secuencia 450: org/w3c/dom/ls/LSEvent.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom.ls;
43
44 import org.w3c.dom.Document;
45 import org.w3c.dom.events.Event;
46
47 /**
```

```
48 * This interface represents a load event object that signals the completion
49 * of a document load.
50 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407'>Document Object
    Model (DOM) Level 3 Load
51 and Save Specification</a>.
52 */
53 public interface LSLoadEvent extends Event {
54     /**
55      * The document that finished loading.
56      */
57     public Document getNewDocument();
58
59     /**
60      * The input source that was parsed.
61      */
62     public LSInput getInput();
63
64 }
```

## 41.5 LSOutput.java

Secuencia 451: org/w3c/dom/ls/LSOutput.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
```

```

32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom.ls;
43
44  /**
45   * This interface represents an output destination for data.
46   * <p> This interface allows an application to encapsulate information about
47   * an output destination in a single object, which may include a URI, a byte
48   * stream (possibly with a specified encoding), a base URI, and/or a
49   * character stream.
50   * <p> The exact definitions of a byte stream and a character stream are
51   * binding dependent.
52   * <p> The application is expected to provide objects that implement this
53   * interface whenever such objects are needed. The application can either
54   * provide its own objects that implement this interface, or it can use the
55   * generic factory method DOMImplementationLS.createLSOutput()
56   * to create objects that implement this interface.
57   * <p> The LSSerializer will use the LSOutput object
58   * to determine where to serialize the output to. The
59   * LSSerializer will look at the different outputs specified in
60   * the LSOutput in the following order to know which one to
61   * output to, the first one that is not null and not an empty string will be
62   * used:
63   * <ol>
64   * <li> LSOutput.characterStream
65   * </li>
66   * <li>
67   * <code>LSOutput.byteStream</code>
68   * </li>
69   * <li> LSOutput.systemId
70   * </li>
71   * </ol>
72   * <p> LSOutput objects belong to the application. The DOM
73   * implementation will never modify them (though it may make copies and
74   * modify the copies, if necessary).
75   * <p> See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407 Document Object
76   * Model (DOM) Level 3 Load
77   * and Save Specification</a>.
78   */
79  public interface LSOutput {
80      /**
81       * An attribute of a language and binding dependent type that represents
82       * a writable stream to which 16-bit units can be output.
83       */
84      public java.io.Writer getCharacterStream();

```

```

84  /**
85   * An attribute of a language and binding dependent type that represents
86   * a writable stream to which 16-bit units can be output.
87   */
88  public void setCharacterStream(java.io.Writer characterStream);
89
90  /**
91   * An attribute of a language and binding dependent type that represents
92   * a writable stream of bytes.
93   */
94  public java.io.OutputStream getByteStream();
95  /**
96   * An attribute of a language and binding dependent type that represents
97   * a writable stream of bytes.
98   */
99  public void setByteStream(java.io.OutputStream byteStream);
100
101  /**
102   * The system identifier, a URI reference [

```



```
130      * Assigned Numbers Authority [131      * should be referred to using their registered names.  
132      */  
133      public void setEncoding(String encoding);  
134  
135  }
```

## 41.6 LSParser.java

Secuencia 452: org/w3c/dom/ls/LSParser.java

```
1  /*  
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.  
3  *  
4  *  
5  *  
6  *  
7  *  
8  *  
9  *  
10 *  
11 *  
12 *  
13 *  
14 *  
15 *  
16 *  
17 *  
18 *  
19 *  
20 *  
21 *  
22 *  
23 */  
24  
25 /*  
26 *  
27 *  
28 *  
29 *  
30 *  
31 * Copyright (c) 2004 World Wide Web Consortium,  
32 *  
33 * (Massachusetts Institute of Technology, European Research Consortium for  
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This  
35 * work is distributed under the W3C(r) Software License [1] in the hope that  
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied  
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
38 *  
39 * [1]   
40 \*/  
41  
42 package org.w3c.dom.ls;
```

```

43
44 import org.w3c.dom.Document;
45 import org.w3c.dom.DOMConfiguration;
46 import org.w3c.dom.Node;
47 import org.w3c.dom.DOMException;
48
49 /**
50  * An interface to an object that is able to build, or augment, a DOM tree
51  * from various input sources.
52  * <p> <code>LSParser</code> provides an API for parsing XML and building the
53  * corresponding DOM document structure. A <code>LSParser</code> instance
54  * can be obtained by invoking the
55  * <code>DOMImplementationLS.createLSParser()</code> method.
56  * <p> As specified in [NODE\_TEXT, and there will never be
62  \* empty text nodes.
63  \* </li>
64  \* <li> it is expected that the <code>value</code> and
65  \* <code>nodeValue</code> attributes of an <code>Attr</code> node initially
66  \* return the <a href='http://www.w3.org/TR/2004/REC-xml-20040204#AVNormalize'>XML 1.0
67  \* normalized value</a>. However, if the parameters "<a href='http://www.w3.org/TR/DOM-Level-3-Core/
68    /core.html#parameter-validate-if-schema'>
69    validate-if-schema</a>" and "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-
70    datatype-normalization'>
71    datatype-normalization</a>" are set to <code>>true</code>, depending on the attribute
72    normalization
73    used, the attribute values may differ from the ones obtained by the XML
74    1.0 attribute normalization. If the parameters "<a href='http://www.w3.org/TR/DOM-Level-3-Core/
75    core.html#parameter-datatype-normalization'>
76    datatype-normalization</a>" is set to <code>>false</code>, the XML 1.0 attribute normalization is
77    guaranteed to occur, and if the attributes list does not contain
78    namespace declarations, the <code>attributes</code> attribute on
79    <code>Element</code> node represents the property <b>\[attributes\]</b> defined in \[

```

```

90 * <code>LSParser</code> signals progress as data is parsed. This
91 * specification does not attempt to define exactly when progress events
92 * should be dispatched. That is intentionally left as
93 * implementation-dependent. Here is one example of how an application might
94 * dispatch progress events: Once the parser starts receiving data, a
95 * progress event is dispatched to indicate that the parsing starts. From
96 * there on, a progress event is dispatched for every 4096 bytes of data
97 * that is received and processed. This is only one example, though, and
98 * implementations can choose to dispatch progress events at any time while
99 * parsing, or not dispatch them at all. See also the definition of the
100 * <code>LSProgressEvent</code> interface. </dd>
101 * </dl>
102 * <p><b>Note:</b> All events defined in this specification use the
103 * namespace URI <code>"http://www.w3.org/2002/DOMLS"</code>.
104 * <p> While parsing an input source, errors are reported to the application
105 * through the error handler (<code>LSParser.domConfig</code>'s "<a href='http://www.w3.org/TR/DOM-
    Level-3-Core/core.html#parameter-error-handler'>
106 * error-handler</a>" parameter). This specification does in no way try to define all possible
107 * errors that can occur while parsing XML, or any other markup, but some
108 * common error cases are defined. The types (<code>DOMError.type</code>) of
109 * errors and warnings defined by this specification are:
110 * <dl>
111 * <dt>
112 * <code>"check-character-normalization-failure" [error]</code> </dt>
113 * <dd> Raised if
114 * the parameter "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-check-
    character-normalization'>
115 * check-character-normalization</a>" is set to true and a string is encountered that fails
    normalization
116 * checking. </dd>
117 * <dt><code>"doctype-not-allowed" [fatal]</code></dt>
118 * <dd> Raised if the
119 * configuration parameter "disallow-doctype" is set to <code>true</code>
120 * and a doctype is encountered. </dd>
121 * <dt><code>"no-input-specified" [fatal]</code></dt>
122 * <dd>
123 * Raised when loading a document and no input is specified in the
124 * <code>LSInput</code> object. </dd>
125 * <dt>
126 * <code>"pi-base-uri-not-preserved" [warning]</code></dt>
127 * <dd> Raised if a processing
128 * instruction is encountered in a location where the base URI of the
129 * processing instruction can not be preserved. One example of a case where
130 * this warning will be raised is if the configuration parameter "<a href='http://www.w3.org/TR/DOM
    -Level-3-Core/core.html#parameter-entities'>
131 * entities</a>" is set to <code>>false</code> and the following XML file is parsed:
132 * <pre>
133 * &lt;!DOCTYPE root [ &lt;!ENTITY e SYSTEM 'subdir/myentity.ent' ]&gt;
134 * &lt;root&gt; &amp;e; &lt;/root&gt;</pre>
135 * And <code>subdir/myentity.ent</code>
136 * contains:
137 * <pre>&lt;one&gt; &lt;two&gt; &lt;/one&gt; &lt;?pi 3.14159?&gt;
138 * &lt;more/&gt;</pre>

```

```

139 * </dd>
140 * <dt><code>"unbound-prefix-in-entity" [warning]</code></dt>
141 * <dd> An
142 * implementation dependent warning that may be raised if the configuration
143 * parameter "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-namespaces'>
144 * namespaces</a>" is set to <code>>true</code> and an unbound namespace prefix is
145 * encountered in an entity's replacement text. Raising this warning is not
146 * enforced since some existing parsers may not recognize unbound namespace
147 * prefixes in the replacement text of entities. </dd>
148 * <dt>
149 * <code>"unknown-character-denormalization" [fatal]</code></dt>
150 * <dd> Raised if the
151 * configuration parameter "ignore-unknown-character-denormalizations" is
152 * set to <code>>false</code> and a character is encountered for which the
153 * processor cannot determine the normalization properties. </dd>
154 * <dt>
155 * <code>"unsupported-encoding" [fatal]</code></dt>
156 * <dd> Raised if an unsupported
157 * encoding is encountered. </dd>
158 * <dt><code>"unsupported-media-type" [fatal]</code></dt>
159 * <dd>
160 * Raised if the configuration parameter "supported-media-types-only" is set
161 * to <code>>true</code> and an unsupported media type is encountered. </dd>
162 * </dl>
163 * <p> In addition to raising the defined errors and warnings, implementations
164 * are expected to raise implementation specific errors and warnings for any
165 * other error and warning cases such as IO errors (file not found,
166 * permission denied,...), XML well-formedness errors, and so on.
167 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407'>Document Object
    Model (DOM) Level 3 Load
168 and Save Specification</a>.
169 */
170 public interface LSParser {
171     /**
172     * The <code>DOMConfiguration</code> object used when parsing an input
173     * source. This <code>DOMConfiguration</code> is specific to the parse
174     * operation. No parameter values from this <code>DOMConfiguration</code>
175     * object are passed automatically to the <code>DOMConfiguration</code>
176     * object on the <code>Document</code> that is created, or used, by the
177     * parse operation. The DOM application is responsible for passing any
178     * needed parameter values from this <code>DOMConfiguration</code>
179     * object to the <code>DOMConfiguration</code> object referenced by the
180     * <code>Document</code> object.
181     * <br> In addition to the parameters recognized in on the <a href='http://www.w3.org/TR/DOM-
        Level-3-Core/core.html#DOMConfiguration'>
182     * DOMConfiguration</a> interface defined in [<a href='http://www.w3.org/TR/2004/REC-DOM-Level
        -3-Core-20040407'>DOM Level 3 Core</a>]
183     * , the <code>DOMConfiguration</code> objects for <code>LSParser</code>
184     * add or modify the following parameters:
185     * <dl>
186     * <dt>
187     * <code>"charset-overrides-xml-encoding"</code></dt>
188     * <dd>

```

```

189 * <dl>
190 * <dt><code>true</code></dt>
191 * <dd><em>optional</em> (<em>default</em>) If a higher level protocol such as HTTP [false</code></dt>
199 \* <dd><em>required</em> The parser ignores any character set encoding information from
200 \* higher-level protocols. </dd>
201 \* </dl></dd>
202 \* <dt><code>"disallow-doctype"</code></dt>
203 \* <dd>
204 \* <dl>
205 \* <dt>
206 \* <code>true</code></dt>
207 \* <dd><em>optional</em> Throw a fatal <b>"doctype-not-allowed"</b> error if a doctype node
    is found while parsing the document. This is
208 \* useful when dealing with things like SOAP envelopes where doctype
209 \* nodes are not allowed. </dd>
210 \* <dt><code>>false</code></dt>
211 \* <dd><em>required</em> \(<em>default</em>\) Allow doctype nodes in the document. </dd>
212 \* </dl></dd>
213 \* <dt>
214 \* <code>"ignore-unknown-character-denormalizations"</code></dt>
215 \* <dd>
216 \* <dl>
217 \* <dt>
218 \* <code>true</code></dt>
219 \* <dd><em>required</em> \(<em>default</em>\) If, while verifying full normalization when \[false</code></dt>
226 \\* <dd><em>optional</em> Report an fatal <b>"unknown-character-denormalization"</b> error if
    a character is encountered for which the processor cannot
227 \\* determine the normalization properties. </dd>
228 \\* </dl></dd>
229 \\* <dt><code>"infoset"</code></dt>
230 \\* <dd> See
231 \\* the definition of <code>DOMConfiguration</code> for a description of
232 \\* this parameter. Unlike in \\[

```

```

235 * <dt><code>"namespaces"</code></dt>
236 * <dd>
237 * <dl>
238 * <dt><code>true</code></dt>
239 * <dd><em>required</em> (<em>default</em>) Perform the namespace processing as defined in [<
    a href='http://www.w3.org/TR/1999/REC-xml-names-19990114/'>XML Namespaces</a>]
240 * and [<a href='http://www.w3.org/TR/2004/REC-xml-names11-20040204/'>XML Namespaces 1.1</a>]
241 * . </dd>
242 * <dt><code>false</code></dt>
243 * <dd><em>optional</em> Do not perform the namespace processing. </dd>
244 * </dl></dd>
245 * <dt>
246 * <code>"resource-resolver"</code></dt>
247 * <dd><em>required</em> A reference to a <code>LSResourceResolver</code> object, or null. If
248 * the value of this parameter is not null when an external resource
249 * (such as an external XML entity or an XML schema location) is
250 * encountered, the implementation will request that the
251 * <code>LSResourceResolver</code> referenced in this parameter resolves
252 * the resource. </dd>
253 * <dt><code>"supported-media-types-only"</code></dt>
254 * <dd>
255 * <dl>
256 * <dt>
257 * <code>true</code></dt>
258 * <dd><em>optional</em> Check that the media type of the parsed resource is a supported
    media
259 * type. If an unsupported media type is encountered, a fatal error of
260 * type <b>"unsupported-media-type"</b> will be raised. The media types defined in [<a href='
    http://www.ietf.org/rfc/rfc3023.txt'>IETF RFC 3023</a>] must always
261 * be accepted. </dd>
262 * <dt><code>false</code></dt>
263 * <dd><em>required</em> (<em>default</em>) Accept any media type. </dd>
264 * </dl></dd>
265 * <dt><code>"validate"</code></dt>
266 * <dd> See the definition of
267 * <code>DOMConfiguration</code> for a description of this parameter.
268 * Unlike in [<a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>DOM Level 3
    Core</a>]
269 * , the processing of the internal subset is always accomplished, even
270 * if this parameter is set to <code>false</code>. </dd>
271 * <dt>
272 * <code>"validate-if-schema"</code></dt>
273 * <dd> See the definition of
274 * <code>DOMConfiguration</code> for a description of this parameter.
275 * Unlike in [<a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>DOM Level 3
    Core</a>]
276 * , the processing of the internal subset is always accomplished, even
277 * if this parameter is set to <code>false</code>. </dd>
278 * <dt>
279 * <code>"well-formed"</code></dt>
280 * <dd> See the definition of
281 * <code>DOMConfiguration</code> for a description of this parameter.
282 * Unlike in [<a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>DOM Level 3

```

```

    Core</a>]
283     * , this parameter cannot be set to <code>>false</code>. </dd>
284     * </dl>
285     */
286     public DOMConfiguration getDomConfig();
287
288     /**
289     * When a filter is provided, the implementation will call out to the
290     * filter as it is constructing the DOM tree structure. The filter can
291     * choose to remove elements from the document being constructed, or to
292     * terminate the parsing early.
293     * <br> The filter is invoked after the operations requested by the
294     * <code>DOMConfiguration</code> parameters have been applied. For
295     * example, if "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-validate'>
296     * validate</a>" is set to <code>>true</code>, the validation is done before invoking the
297     * filter.
298     */
299     public LSParserFilter getFilter();
300
301     /**
302     * When a filter is provided, the implementation will call out to the
303     * filter as it is constructing the DOM tree structure. The filter can
304     * choose to remove elements from the document being constructed, or to
305     * terminate the parsing early.
306     * <br> The filter is invoked after the operations requested by the
307     * <code>DOMConfiguration</code> parameters have been applied. For
308     * example, if "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-validate'>
309     * validate</a>" is set to <code>>true</code>, the validation is done before invoking the
310     * filter.
311     */
312     public void setFilter(LSParserFilter filter);
313
314     /**
315     * <code>true</code> if the <code>LSParser</code> is asynchronous,
316     * <code>false</code> if it is synchronous.
317     */
318     public boolean getAsync();
319
320     /**
321     * <code>true</code> if the <code>LSParser</code> is currently busy
322     * loading a document, otherwise <code>false</code>.
323     */
324     public boolean getBusy();
325
326     /**
327     * Parse an XML document from a resource identified by a
328     * <code>LSInput</code>.
329     * @param input The <code>LSInput</code> from which the source of the
330     * document is to be read.
331     * @return If the <code>LSParser</code> is a synchronous
332     * <code>LSParser</code>, the newly created and populated
333     * <code>Document</code> is returned. If the <code>LSParser</code> is
334     * asynchronous, <code>null</code> is returned since the document
335     * object may not yet be constructed when this method returns.

```



```

335     * @exception DOMException
336     *     INVALID_STATE_ERR: Raised if the <code>LSParser</code>'s
337     *     <code>LSParser.busy</code> attribute is <code>>true</code>.
338     * @exception LSEException
339     *     PARSE_ERR: Raised if the <code>LSParser</code> was unable to load
340     *     the XML document. DOM applications should attach a
341     *     <code>DOMErrorHandler</code> using the parameter "<a href='http://www.w3.org/TR/DOM-Level
342     *     -3-Core/core.html#parameter-error-handler'>
343     *     error-handler</a>" if they wish to get details on the error.
344     */
345     public Document parse(LSInput input)
346         throws DOMException, LSEException;
347
348     /**
349     * Parse an XML document from a location identified by a URI reference [<a href='http://www.
350     * ietf.org/rfc/rfc2396.txt'>IETF RFC 2396</a>]. If the URI
351     * contains a fragment identifier (see section 4.1 in [<a href='http://www.ietf.org/rfc/rfc2396
352     * .txt'>IETF RFC 2396</a>]), the
353     * behavior is not defined by this specification, future versions of
354     * this specification may define the behavior.
355     * @param uri The location of the XML document to be read.
356     * @return If the <code>LSParser</code> is a synchronous
357     * <code>LSParser</code>, the newly created and populated
358     * <code>Document</code> is returned, or <code>null</code> if an error
359     * occurred. If the <code>LSParser</code> is asynchronous,
360     * <code>null</code> is returned since the document object may not yet
361     * be constructed when this method returns.
362     * @exception DOMException
363     *     INVALID_STATE_ERR: Raised if the <code>LSParser.busy</code>
364     *     attribute is <code>>true</code>.
365     * @exception LSEException
366     *     PARSE_ERR: Raised if the <code>LSParser</code> was unable to load
367     *     the XML document. DOM applications should attach a
368     *     <code>DOMErrorHandler</code> using the parameter "<a href='http://www.w3.org/TR/DOM-Level
369     *     -3-Core/core.html#parameter-error-handler'>
370     *     error-handler</a>" if they wish to get details on the error.
371     */
372     public Document parseURI(String uri)
373         throws DOMException, LSEException;
374
375     // ACTION_TYPES
376
377     /**
378     * Append the result of the parse operation as children of the context
379     * node. For this action to work, the context node must be an
380     * <code>Element</code> or a <code>DocumentFragment</code>.
381     */
382     public static final short ACTION_APPEND_AS_CHILDREN = 1;
383
384     /**
385     * Replace all the children of the context node with the result of the
386     * parse operation. For this action to work, the context node must be an
387     * <code>Element</code>, a <code>Document</code>, or a
388     * <code>DocumentFragment</code>.
389     */

```



```

384 public static final short ACTION_REPLACE_CHILDREN    = 2;
385 /**
386  * Insert the result of the parse operation as the immediately preceding
387  * sibling of the context node. For this action to work the context
388  * node's parent must be an Element or a
389  * DocumentFragment.
390  */
391 public static final short ACTION_INSERT_BEFORE      = 3;
392 /**
393  * Insert the result of the parse operation as the immediately following
394  * sibling of the context node. For this action to work the context
395  * node's parent must be an Element or a
396  * DocumentFragment.
397  */
398 public static final short ACTION_INSERT_AFTER       = 4;
399 /**
400  * Replace the context node with the result of the parse operation. For
401  * this action to work, the context node must have a parent, and the
402  * parent must be an Element or a
403  * DocumentFragment.
404  */
405 public static final short ACTION_REPLACE           = 5;
406
407 /**
408  * Parse an XML fragment from a resource identified by a
409  * LSInput and insert the content into an existing document
410  * at the position specified with the context and
411  * action arguments. When parsing the input stream, the
412  * context node (or its parent, depending on where the result will be
413  * inserted) is used for resolving unbound namespace prefixes. The
414  * context node's ownerDocument node (or the node itself if
415  * the node of type DOCUMENT_NODE) is used to resolve
416  * default attributes and entity references.
417  *   
 As the new data is inserted into the document, at least one
418  * mutation event is fired per new immediate child or sibling of the
419  * context node.
420  *   
 If the context node is a Document node and the action
421  * is ACTION_REPLACE_CHILDREN, then the document that is
422  * passed as the context node will be changed such that its
423  * xmlEncoding, documentURI,
424  * xmlVersion, inputEncoding,
425  * xmlStandalone, and all other such attributes are set to
426  * what they would be set to if the input source was parsed using
427  * LSParser.parse().
428  *   
 This method is always synchronous, even if the
429  * LSParser is asynchronous (LSParser.async is
430  * true).
431  *   
 If an error occurs while parsing, the caller is notified through
432  * the ErrorHandler instance associated with the ""
435  \*   parameter of the DOMConfiguration.
436  \*   
 When calling parseWithContext, the values of the
437  \* following configuration parameters will be ignored and their default

```

```

436 * values will always be used instead: "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.
      html#parameter-validate'>
437 * validate</a>", "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-validate-
      if-schema'>
438 * validate-if-schema</a>", and "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#
      parameter-element-content-whitespace'>
439 * element-content-whitespace</a>". Other parameters will be treated normally, and the parser
      is expected
440 * to call the <code>LSParserFilter</code> just as if a whole document
441 * was parsed.
442 * @param input The <code>LSInput</code> from which the source document
443 * is to be read. The source document must be an XML fragment, i.e.
444 * anything except a complete XML document (except in the case where
445 * the context node of type <code>DOCUMENT_NODE</code>, and the action
446 * is <code>ACTION_REPLACE_CHILDREN</code>), a DOCTYPE (internal
447 * subset), entity declaration(s), notation declaration(s), or XML or
448 * text declaration(s).
449 * @param contextArg The node that is used as the context for the data
450 * that is being parsed. This node must be a <code>Document</code>
451 * node, a <code>DocumentFragment</code> node, or a node of a type
452 * that is allowed as a child of an <code>Element</code> node, e.g. it
453 * cannot be an <code>Attribute</code> node.
454 * @param action This parameter describes which action should be taken
455 * between the new set of nodes being inserted and the existing
456 * children of the context node. The set of possible actions is
457 * defined in <code>ACTION_TYPES</code> above.
458 * @return Return the node that is the result of the parse operation. If
459 * the result is more than one top-level node, the first one is
460 * returned.
461 * @exception DOMException
462 * HIERARCHY_REQUEST_ERR: Raised if the content cannot replace, be
463 * inserted before, after, or as a child of the context node (see also
464 * <code>Node.insertBefore</code> or <code>Node.replaceChild</code> in [

```

```
483     public Node parseWithContext(LSInput input,
484                                 Node contextArg,
485                                 short action)
486         throws DOMException, LSEException;
487
488     /**
489     * Abort the loading of the document that is currently being loaded by
490     * the <code>LSParser</code>. If the <code>LSParser</code> is currently
491     * not busy, a call to this method does nothing.
492     */
493     public void abort();
494
495 }
```

## 41.7 LSParserFilter.java

Secuencia 453: [org.w3c/dom/ls/LSParserFilter.java](http://org.w3c/dom/ls/LSParserFilter.java)

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
```

```

37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom.ls;
43
44  import org.w3c.dom.Node;
45  import org.w3c.dom.Element;
46
47  /**
48   * <code>LSParserFilter</code>s provide applications the ability to examine
49   * nodes as they are being constructed while parsing. As each node is
50   * examined, it may be modified or removed, or the entire parse may be
51   * terminated early.
52   * <p> At the time any of the filter methods are called by the parser, the
53   * owner Document and DOMImplementation objects exist and are accessible.
54   * The document element is never passed to the <code>LSParserFilter</code>
55   * methods, i.e. it is not possible to filter out the document element.
56   * <code>Document</code>, <code>DocumentType</code>, <code>Notation</code>,
57   * <code>Entity</code>, and <code>Attr</code> nodes are never passed to the
58   * <code>acceptNode</code> method on the filter. The child nodes of an
59   * <code>EntityReference</code> node are passed to the filter if the
60   * parameter "<a href='http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-entities'>
61   * entities</a>" is set to <code>>false</code>. Note that, as described by the parameter "<a href='
62   * http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-entities>
63   * entities</a>", unexpanded entity reference nodes are never discarded and are always
64   * passed to the filter.
65   * <p> All validity checking while parsing a document occurs on the source
66   * document as it appears on the input stream, not on the DOM document as it
67   * is built in memory. With filters, the document in memory may be a subset
68   * of the document on the stream, and its validity may have been affected by
69   * the filtering.
70   * <p> All default attributes must be present on elements when the elements
71   * are passed to the filter methods. All other default content must be
72   * passed to the filter methods.
73   * <p> DOM applications must not raise exceptions in a filter. The effect of
74   * throwing exceptions from a filter is DOM implementation dependent.
75   * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407'>Document Object
76   * Model (DOM) Level 3 Load
77   * and Save Specification</a>.
78   */
79  public interface LSParserFilter {
80      // Constants returned by startElement and acceptNode
81      /**
82       * Accept the node.
83       */
84      public static final short FILTER_ACCEPT = 1;
85      /**
86       * Reject the node and its children.
87       */
88      public static final short FILTER_REJECT = 2;
89  }

```

```

88     * Skip this single node. The children of this node will still be
89     * considered.
90     */
91     public static final short FILTER_SKIP                = 3;
92     /**
93     * Interrupt the normal processing of the document.
94     */
95     public static final short FILTER_INTERRUPT           = 4;
96
97     /**
98     * The parser will call this method after each <code>Element</code> start
99     * tag has been scanned, but before the remainder of the
100    * <code>Element</code> is processed. The intent is to allow the
101    * element, including any children, to be efficiently skipped. Note that
102    * only element nodes are passed to the <code>startElement</code>
103    * function.
104    * <br>The element node passed to <code>startElement</code> for filtering
105    * will include all of the Element's attributes, but none of the
106    * children nodes. The Element may not yet be in place in the document
107    * being constructed (it may not have a parent node.)
108    * <br>A <code>startElement</code> filter function may access or change
109    * the attributes for the Element. Changing Namespace declarations will
110    * have no effect on namespace resolution by the parser.
111    * <br>For efficiency, the Element node passed to the filter may not be
112    * the same one as is actually placed in the tree if the node is
113    * accepted. And the actual node (node object identity) may be reused
114    * during the process of reading in and filtering a document.
115    * @param elementArg The newly encountered element. At the time this
116    *     method is called, the element is incomplete - it will have its
117    *     attributes, but no children.
118    * @return
119    *     <ul>
120    *     <li> <code>FILTER_ACCEPT</code> if the <code>Element</code> should
121    *     be included in the DOM document being built.
122    *     </li>
123    *     <li>
124    *     <code>FILTER_REJECT</code> if the <code>Element</code> and all of
125    *     its children should be rejected.
126    *     </li>
127    *     <li> <code>FILTER_SKIP</code> if the
128    *     <code>Element</code> should be skipped. All of its children are
129    *     inserted in place of the skipped <code>Element</code> node.
130    *     </li>
131    *     <li>
132    *     <code>FILTER_INTERRUPT</code> if the filter wants to stop the
133    *     processing of the document. Interrupting the processing of the
134    *     document does no longer guarantee that the resulting DOM tree is
135    *     XML well-formed. The <code>Element</code> is rejected.
136    *     </li>
137    *     </ul> Returning
138    *     any other values will result in unspecified behavior.
139    */
140     public short startElement(Element elementArg);

```

```

141
142 /**
143  * This method will be called by the parser at the completion of the
144  * parsing of each node. The node and all of its descendants will exist
145  * and be complete. The parent node will also exist, although it may be
146  * incomplete, i.e. it may have additional children that have not yet
147  * been parsed. Attribute nodes are never passed to this function.
148  * <br>From within this method, the new node may be freely modified -
149  * children may be added or removed, text nodes modified, etc. The state
150  * of the rest of the document outside this node is not defined, and the
151  * affect of any attempt to navigate to, or to modify any other part of
152  * the document is undefined.
153  * <br>For validating parsers, the checks are made on the original
154  * document, before any modification by the filter. No validity checks
155  * are made on any document modifications made by the filter.
156  * <br>If this new node is rejected, the parser might reuse the new node
157  * and any of its descendants.
158  * @param nodeArg The newly constructed element. At the time this method
159  *   is called, the element is complete - it has all of its children
160  *   (and their children, recursively) and attributes, and is attached
161  *   as a child to its parent.
162  * @return
163  *   <ul>
164  *   <li> <code>FILTER_ACCEPT</code> if this <code>Node</code> should
165  *   be included in the DOM document being built.
166  *   </li>
167  *   <li>
168  *   <code>FILTER_REJECT</code> if the <code>Node</code> and all of its
169  *   children should be rejected.
170  *   </li>
171  *   <li> <code>FILTER_SKIP</code> if the
172  *   <code>Node</code> should be skipped and the <code>Node</code>
173  *   should be replaced by all the children of the <code>Node</code>.
174  *   </li>
175  *   <li>
176  *   <code>FILTER_INTERRUPT</code> if the filter wants to stop the
177  *   processing of the document. Interrupting the processing of the
178  *   document does no longer guarantee that the resulting DOM tree is
179  *   XML well-formed. The <code>Node</code> is accepted and will be the
180  *   last completely parsed node.
181  *   </li>
182  *   </ul>
183  */
184 public short acceptNode(Node nodeArg);
185
186 /**
187  * Tells the <code>LSParser</code> what types of nodes to show to the
188  * method <code>LSParserFilter.acceptNode</code>. If a node is not shown
189  * to the filter using this attribute, it is automatically included in
190  * the DOM document being built. See <code>NodeFilter</code> for
191  * definition of the constants. The constants <code>SHOW_ATTRIBUTE</code>
192  * , <code>SHOW_DOCUMENT</code>, <code>SHOW_DOCUMENT_TYPE</code>,
193  * <code>SHOW_NOTATION</code>, <code>SHOW_ENTITY</code>, and

```

```

194     * <code>SHOW_DOCUMENT_FRAGMENT</code> are meaningless here. Those nodes
195     * will never be passed to <code>LSParserFilter.acceptNode</code>.
196     * <br> The constants used here are defined in [

```

## 41.8 LSProgressEvent.java

Secuencia 454: org/w3c/dom/ls/LSProgressEvent.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] 
40 \*/

```

```

41
42 package org.w3c.dom.ls;
43
44 import org.w3c.dom.events.Event;
45
46 /**
47  * This interface represents a progress event object that notifies the
48  * application about progress as a document is parsed. It extends the
49  * Event interface defined in [position and
52  \* totalSize are not specified and can be implementation and
53  \* input dependent.
54  \* <p>See also the 0 is
74      \\* returned if the total size cannot be determined or estimated.
75      \\*/
76     public int getTotalSize\\(\\);
77 }

```

## 41.9 LSResourceResolver.java

Secuencia 455: org/w3c/dom/ls/LSResourceResolver.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```



```

11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42  package org.w3c.dom.ls;
43
44  /**
45   * <code>LSResourceResolver</code> provides a way for applications to
46   * redirect references to external resources.
47   * <p> Applications needing to implement custom handling for external
48   * resources can implement this interface and register their implementation
49   * by setting the "resource-resolver" parameter of
50   * <code>DOMConfiguration</code> objects attached to <code>LSParser</code>
51   * and <code>LSSerializer</code>. It can also be register on
52   * <code>DOMConfiguration</code> objects attached to <code>Document</code>
53   * if the "LS" feature is supported.
54   * <p> The <code>LSParser</code> will then allow the application to intercept
55   * any external entities, including the external DTD subset and external
56   * parameter entities, before including them. The top-level document entity
57   * is never passed to the <code>resolveResource</code> method.
58   * <p> Many DOM applications will not need to implement this interface, but it
59   * will be especially useful for applications that build XML documents from
60   * databases or other specialized input sources, or for applications that
61   * use URNs.
62   * <p> <b>Note:</b> <code>LSResourceResolver</code> is based on the SAX2 [http://www.
        saxproject.org/>SAX</a>] <code>EntityResolver</code>

```



**41.10 LSSerializer.java**

Secuencia 456: org/w3c/dom/ls/LSSerializer.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2004 World Wide Web Consortium,
32 *
33 * (Massachusetts Institute of Technology, European Research Consortium for
34 * Informatics and Mathematics, Keio University). All Rights Reserved. This
35 * work is distributed under the W3C(r) Software License [1] in the hope that
36 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38 *
39 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40 */
41
42 package org.w3c.dom.ls;
43
44 import org.w3c.dom.DOMConfiguration;
45 import org.w3c.dom.Node;
46 import org.w3c.dom.DOMException;
47
48 /**
49 * A LSSerializer provides an API for serializing (writing) a
50 * DOM document out into XML. The XML data is written to a string or an
```

```

51 * output stream. Any changes or fixups made during the serialization affect
52 * only the serialized data. The Document object and its
53 * children are never altered by the serialization operation.
54 * 

During serialization of XML data, namespace fixup is done as defined in [\]
55 \* , Appendix B. \[\\]
56 \\* allows empty strings as a real namespace URI. If the
57 \\* namespaceURI of a Node is empty string, the
58 \\* serialization will treat them as null, ignoring the prefix
59 \\* if any.
60 \\* 

LSSerializer accepts any node type for serialization. For
61 \\* nodes of type Document or Entity, well-formed
62 \\* XML will be created when possible \\(well-formedness is guaranteed if the
63 \\* document or entity comes from a parse operation and is unchanged since it
64 \\* was created\\). The serialized output for these node types is either as a
65 \\* XML document or an External XML Entity, respectively, and is acceptable
66 \\* input for an XML parser. For all other types of nodes the serialized form
67 \\* is implementation dependent.
68 \\* 

Within a Document, DocumentFragment, or
69 \\* Entity being serialized, Nodes are processed as
70 \\* follows
71 \\* 


72 \\* - Document nodes are written, including the XML
73 \\* declaration \\(unless the parameter "xml-declaration" is set to
74 \\* false\\) and a DTD subset, if one exists in the DOM. Writing a
75 \\* Document node serializes the entire document.
76 \\*

77 \\* - 78 \\* Entity nodes, when written directly by
79 \\* LSSerializer.write, outputs the entity expansion but no
80 \\* namespace fixup is done. The resulting output will be valid as an
81 \\* external entity.
82 \\*

83 \\* - If the parameter "" is set to true, EntityReference nodes are
84 \\\* serialized as an entity reference of the form "
85 \\\* &entityName;" in the output. Child nodes \\\(the expansion\\\)
86 \\\* of the entity reference are ignored. If the parameter "" is set to false, only the children of the entity reference
87 \\\\* are serialized. EntityReference nodes with no children \\\\(no
88 \\\\* corresponding Entity node or the corresponding
89 \\\\* Entity nodes have no children\\\\) are always serialized.
90 \\\\*

91 \\* - 92 \\* CDATAsections containing content characters that cannot be
93 \\* represented in the specified output encoding are handled according to the
94 \\* "" parameter. If the parameter is set to true,
95 \\\* CDATAsections are split, and the unrepresentable characters
96 \\\* are serialized as numeric character references in ordinary content. The


```

```

100 * exact position and number of splits is not specified. If the parameter
101 * is set to <code>false</code>, unrepresentable characters in a
102 * <code>CDATASection</code> are reported as
103 * <code>"wf-invalid-character"</code> errors if the parameter "<code>true</code>. The error is not recoverable - there is no
105 \* mechanism for supplying alternative characters and continuing with the
106 \* serialization.
107 \* </li>
108 \* <li> <code>DocumentFragment</code> nodes are serialized by
109 \* serializing the children of the document fragment in the order they
110 \* appear in the document fragment.
111 \* </li>
112 \* <li> All other node types \(Element, Text,
113 \* etc.\) are serialized to their corresponding XML source form.
114 \* </li>
115 \* </ul>
116 \* <p><b>Note:</b> The serialization of a <code>Node</code> does not always
117 \* generate a well-formed XML document, i.e. a <code>LSParser</code> might
118 \* throw fatal errors when parsing the resulting serialization.
119 \* <p> Within the character data of a document \(outside of markup\), any
120 \* characters that cannot be represented directly are replaced with
121 \* character references. Occurrences of '&lt;' and '&amp;' are replaced by
122 \* the predefined entities &amp;lt; and &amp;amp;. The other predefined
123 \* entities \(&amp;gt;, &amp;apos;, and &amp;quot;;\) might not be used, except
124 \* where needed \(e.g. using &amp;gt; in cases such as '\]\]&gt;'\). Any
125 \* characters that cannot be represented directly in the output character
126 \* encoding are serialized as numeric character references \(and since
127 \* character encoding standards commonly use hexadecimal representations of
128 \* characters, using the hexadecimal representation when serializing
129 \* character references is encouraged\).
130 \* <p> To allow attribute values to contain both single and double quotes, the
131 \* apostrophe or single-quote character \('\) may be represented as
132 \* "&amp;apos;", and the double-quote character \("\) as "&amp;quot;". New
133 \* line characters and other characters that cannot be represented directly
134 \* in attribute values in the output character encoding are serialized as a
135 \* numeric character reference.
136 \* <p> Within markup, but outside of attributes, any occurrence of a character
137 \* that cannot be represented in the output character encoding is reported
138 \* as a <code>DOMError</code> fatal error. An example would be serializing
139 \* the element &lt;LaCa\u00f1ada&gt; with <code>encoding="us-ascii"</code>.
140 \* This will result with a generation of a <code>DOMError</code>
141 \* "wf-invalid-character-in-node-name" \(as proposed in "normalize-characters</a>" on <code>LSSerializer</code> to true, character normalization is
145 \\* performed according to the definition of normalized</a> characters included in appendix E of \\\[

```

```

148 * normalization process affects only the data as it is being written; it
149 * does not alter the DOM's view of the document after serialization has
150 * completed.
151 * <p> Implementations are required to support the encodings "UTF-8",
152 * "UTF-16", "UTF-16BE", and "UTF-16LE" to guarantee that data is
153 * serializable in all encodings that are required to be supported by all
154 * XML parsers. When the encoding is UTF-8, whether or not a byte order mark
155 * is serialized, or if the output is big-endian or little-endian, is
156 * implementation dependent. When the encoding is UTF-16, whether or not the
157 * output is big-endian or little-endian is implementation dependent, but a
158 * Byte Order Mark must be generated for non-character outputs, such as
159 * <code>LSOutput.byteStream</code> or <code>LSOutput.systemId</code>. If
160 * the Byte Order Mark is not generated, a "byte-order-mark-needed" warning
161 * is reported. When the encoding is UTF-16LE or UTF-16BE, the output is
162 * big-endian (UTF-16BE) or little-endian (UTF-16LE) and the Byte Order Mark
163 * is not be generated. In all cases, the encoding declaration, if
164 * generated, will correspond to the encoding used during the serialization
165 * (e.g. <code>encoding="UTF-16"</code> will appear if UTF-16 was
166 * requested).
167 * <p> Namespaces are fixed up during serialization, the serialization process
168 * will verify that namespace declarations, namespace prefixes and the
169 * namespace URI associated with elements and attributes are consistent. If
170 * inconsistencies are found, the serialized form of the document will be
171 * altered to remove them. The method used for doing the namespace fixup
172 * while serializing a document is the algorithm defined in Appendix B.1,
173 * "Namespace normalization", of [

```

```

198 * <dd> Raised if an unsupported
199 * encoding is encountered. </dd>
200 * </dl>
201 * <p> In addition to raising the defined errors and warnings, implementations
202 * are expected to raise implementation specific errors and warnings for any
203 * other error and warning cases such as IO errors (file not found,
204 * permission denied,...) and so on.
205 * <p>See also the <a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407'>Document Object
    Model (DOM) Level 3 Load
206 and Save Specification</a>.
207 */
208 public interface LSSerializer {
209     /**
210      * The <code>DOMConfiguration</code> object used by the
211      * <code>LSSerializer</code> when serializing a DOM node.
212      * <br> In addition to the parameters recognized by the <a href='http://www.w3.org/TR/DOM-Level
        -3-Core/core.html#DOMConfiguration'>
213      * DOMConfiguration</a> interface defined in [<a href='http://www.w3.org/TR/2004/REC-DOM-Level
        -3-Core-20040407'>DOM Level 3 Core</a>]
214      * , the <code>DOMConfiguration</code> objects for
215      * <code>LSSerializer</code> adds, or modifies, the following
216      * parameters:
217      * <dl>
218      * <dt><code>"canonical-form"</code></dt>
219      * <dd>
220      * <dl>
221      * <dt><code>true</code></dt>
222      * <dd><em>optional</em> Writes the document according to the rules specified in [<a href='
        http://www.w3.org/TR/2001/REC-xml-c14n-20010315'>Canonical XML</a>].
223      * In addition to the behavior described in "<a href='http://www.w3.org/TR/DOM-Level-3-Core/
        core.html#parameter-canonical-form'>
224      * canonical-form</a>" [<a href='http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407'>DOM
        Level 3 Core</a>]
225      * , setting this parameter to <code>true</code> will set the parameters
226      * "format-pretty-print", "discard-default-content", and "xml-declaration
227      * ", to <code>false</code>. Setting one of those parameters to
228      * <code>true</code> will set this parameter to <code>false</code>.
229      * Serializing an XML 1.1 document when "canonical-form" is
230      * <code>true</code> will generate a fatal error. </dd>
231      * <dt><code>false</code></dt>
232      * <dd><em>required</em> (<em>default</em>) Do not canonicalize the output. </dd>
233      * </dl></dd>
234      * <dt><code>"discard-default-content"</code></dt>
235      * <dd>
236      * <dl>
237      * <dt>
238      * <code>true</code></dt>
239      * <dd><em>required</em> (<em>default</em>) Use the <code>Attr.specified</code> attribute to
        decide what attributes
240      * should be discarded. Note that some implementations might use
241      * whatever information available to the implementation (i.e. XML
242      * schema, DTD, the <code>Attr.specified</code> attribute, and so on) to
243      * determine what attributes and content to discard if this parameter is

```



```

244 * set to true. </dd>
245 * <dt>false</dt>
246 * <dd><em>required</em>Keep all attributes and all content.</dd>
247 * </dl></dd>
248 * <dt>"format-pretty-print"</dt>
249 * <dd>
250 * <dl>
251 * <dt>
252 * <code>true</code></dt>
253 * <dd><em>optional</em> Formatting the output by adding whitespace to produce a pretty-
    printed,
254 * indented, human-readable form. The exact form of the transformations
255 * is not specified by this specification. Pretty-printing changes the
256 * content of the document and may affect the validity of the document,
257 * validating implementations should preserve validity. </dd>
258 * <dt>
259 * <code>>false</code></dt>
260 * <dd><em>required</em> (<em>default</em>) Don't pretty-print the result. </dd>
261 * </dl></dd>
262 * <dt>
263 * <code>"ignore-unknown-character-denormalizations"</code> </dt>
264 * <dd>
265 * <dl>
266 * <dt>
267 * <code>true</code></dt>
268 * <dd><em>required</em> (<em>default</em>) If, while verifying full normalization when [false</code></dt>
276 \* <dd><em>optional</em> Report a fatal error if a character is encountered for which the
277 \* processor cannot determine the normalization properties. </dd>
278 \* </dl></dd>
279 \* <dt>
280 \* <code>"normalize-characters"</code></dt>
281 \* <dd> This parameter is equivalent to
282 \* the one defined by DOMConfiguration in \[

```



```

293 * <dd><em>required</em> (<em>default</em>) If a <code>Document</code>, <code>Element</code>,
    or <code>Entity</code>
294 * node is serialized, the XML declaration, or text declaration, should
295 * be included. The version (<code>Document.xmlVersion</code> if the
296 * document is a Level 3 document and the version is non-null, otherwise
297 * use the value "1.0"), and the output encoding (see
298 * <code>LSSerializer.write</code> for details on how to find the output
299 * encoding) are specified in the serialized XML declaration. </dd>
300 * <dt>
301 * <code>false</code></dt>
302 * <dd><em>required</em> Do not serialize the XML and text declarations. Report a
303 * <code>"xml-declaration-needed"</code> warning if this will cause
304 * problems (i.e. the serialized data is of an XML version other than [

```

```

342 * <br> On retrieval, the default value of this attribute is the
343 * implementation specific default end-of-line sequence. DOM
344 * implementations should choose the default to match the usual
345 * convention for text files in the environment being used.
346 * Implementations must choose a default sequence that matches one of
347 * those allowed by XML 1.0 or XML 1.1, depending on the serialized
348 * content. Setting this attribute to <code>null</code> will reset its
349 * value to the default value.
350 * <br>
351 */
352 public void setNewLine(String newLine);
353
354 /**
355 * When the application provides a filter, the serializer will call out
356 * to the filter before serializing each Node. The filter implementation
357 * can choose to remove the node from the stream or to terminate the
358 * serialization early.
359 * <br> The filter is invoked after the operations requested by the
360 * <code>DOMConfiguration</code> parameters have been applied. For
361 * example, CDATA sections won't be passed to the filter if "<a href='http://www.w3.org/TR/DOM-
    Level-3-Core/core.html#parameter-cdata-sections'>
362 * cdata-sections</a>" is set to <code>>false</code>.
363 */
364 public LSSerializerFilter getFilter();
365 /**
366 * When the application provides a filter, the serializer will call out
367 * to the filter before serializing each Node. The filter implementation
368 * can choose to remove the node from the stream or to terminate the
369 * serialization early.
370 * <br> The filter is invoked after the operations requested by the
371 * <code>DOMConfiguration</code> parameters have been applied. For
372 * example, CDATA sections won't be passed to the filter if "<a href='http://www.w3.org/TR/DOM-
    Level-3-Core/core.html#parameter-cdata-sections'>
373 * cdata-sections</a>" is set to <code>>false</code>.
374 */
375 public void setFilter(LSSerializerFilter filter);
376
377 /**
378 * Serialize the specified node as described above in the general
379 * description of the <code>LSSerializer</code> interface. The output is
380 * written to the supplied <code>LSOutput</code>.
381 * <br> When writing to a <code>LSOutput</code>, the encoding is found by
382 * looking at the encoding information that is reachable through the
383 * <code>LSOutput</code> and the item to be written (or its owner
384 * document) in this order:
385 * <ol>
386 * <li> <code>LSOutput.encoding</code>,
387 * </li>
388 * <li>
389 * <code>Document.inputEncoding</code>,
390 * </li>
391 * <li>
392 * <code>Document.xmlEncoding</code>.

```

```

393     * </li>
394     * </ol>
395     * <br> If no encoding is reachable through the above properties, a
396     * default encoding of "UTF-8" will be used. If the specified encoding
397     * is not supported an "unsupported-encoding" fatal error is raised.
398     * <br> If no output is specified in the <code>LSOutput</code>, a
399     * "no-output-specified" fatal error is raised.
400     * <br> The implementation is responsible of associating the appropriate
401     * media type with the serialized data.
402     * <br> When writing to a HTTP URI, a HTTP PUT is performed. When writing
403     * to other types of URIs, the mechanism for writing the data to the URI
404     * is implementation dependent.
405     * @param nodeArg The node to serialize.
406     * @param destination The destination for the serialized DOM.
407     * @return Returns <code>true</code> if <code>node</code> was
408     * successfully serialized. Return <code>false</code> in case the
409     * normal processing stopped but the implementation kept serializing
410     * the document; the result of the serialization being implementation
411     * dependent then.
412     * @exception LSEException
413     *     SERIALIZE_ERR: Raised if the <code>LSSerializer</code> was unable to
414     *     serialize the node. DOM applications should attach a
415     *     <code>DOMErrorHandler</code> using the parameter "<a href='http://www.w3.org/TR/DOM-Level
416     *     -3-Core/core.html#parameter-error-handler'>
417     *     error-handler</a>" if they wish to get details on the error.
418     */
419     public boolean write(Node nodeArg,
420                          LSOutput destination)
421         throws LSEException;
422
423     /**
424     * A convenience method that acts as if <code>LSSerializer.write</code>
425     * was called with a <code>LSOutput</code> with no encoding specified
426     * and <code>LSOutput.systemId</code> set to the <code>uri</code>
427     * argument.
428     * @param nodeArg The node to serialize.
429     * @param uri The URI to write to.
430     * @return Returns <code>true</code> if <code>node</code> was
431     * successfully serialized. Return <code>false</code> in case the
432     * normal processing stopped but the implementation kept serializing
433     * the document; the result of the serialization being implementation
434     * dependent then.
435     * @exception LSEException
436     *     SERIALIZE_ERR: Raised if the <code>LSSerializer</code> was unable to
437     *     serialize the node. DOM applications should attach a
438     *     <code>DOMErrorHandler</code> using the parameter "<a href='http://www.w3.org/TR/DOM-Level
439     *     -3-Core/core.html#parameter-error-handler'>
440     *     error-handler</a>" if they wish to get details on the error.
441     */
442     public boolean writeToURI(Node nodeArg,
443                               String uri)
444         throws LSEException;
445

```

```

444  /**
445   * Serialize the specified node as described above in the general
446   * description of the <code>LSSerializer</code> interface. The output is
447   * written to a <code>DOMString</code> that is returned to the caller.
448   * The encoding used is the encoding of the <code>DOMString</code> type,
449   * i.e. UTF-16. Note that no Byte Order Mark is generated in a
450   * <code>DOMString</code> object.
451   * @param nodeArg The node to serialize.
452   * @return Returns the serialized data.
453   * @exception DOMException
454   *     DOMSTRING_SIZE_ERR: Raised if the resulting string is too long to
455   *     fit in a <code>DOMString</code>.
456   * @exception LSEException
457   *     SERIALIZE_ERR: Raised if the <code>LSSerializer</code> was unable to
458   *     serialize the node. DOM applications should attach a
459   *     <code>DOMErrorHandler</code> using the parameter "<a href='http://www.w3.org/TR/DOM-Level
460   *     -3-Core/core.html#parameter-error-handler'>
461   *     error-handler</a>" if they wish to get details on the error.
462   */
463   public String writeToString(Node nodeArg)
464       throws DOMException, LSEException;
465 }

```

#### 41.11 LSSerializerFilter.java

Secuencia 457: org/w3c/dom/ls/LSSerializerFilter.java

```

1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *

```

```

27  *
28  *
29  *
30  *
31  * Copyright (c) 2004 World Wide Web Consortium,
32  *
33  * (Massachusetts Institute of Technology, European Research Consortium for
34  * Informatics and Mathematics, Keio University). All Rights Reserved. This
35  * work is distributed under the W3C(r) Software License [1] in the hope that
36  * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
37  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
38  *
39  * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
40  */
41
42 package org.w3c.dom.ls;
43
44 import org.w3c.dom.traversal.NodeFilter;
45
46 /**
47  * <code>LSSerializerFilter</code>s provide applications the ability to
48  * examine nodes as they are being serialized and decide what nodes should
49  * be serialized or not. The <code>LSSerializerFilter</code> interface is
50  * based on the <code>NodeFilter</code> interface defined in [http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113] DOM Level 2 Traversal and Range</a>]
51  * .
52  * <p> <code>Document</code>, <code>DocumentType</code>,
53  * <code>DocumentFragment</code>, <code>Notation</code>, <code>Entity</code>
54  * , and children of <code>Attr</code> nodes are not passed to the filter.
55  * The child nodes of an <code>EntityReference</code> node are only passed
56  * to the filter if the <code>EntityReference</code> node is skipped by the
57  * method <code>LSParserFilter.acceptNode()</code>.
58  * <p> When serializing an <code>Element</code>, the element is passed to the
59  * filter before any of its attributes are passed to the filter. Namespace
60  * declaration attributes, and default attributes (except in the case when "
61  * discard-default-content" is set to <code>>false</code>), are never passed
62  * to the filter.
63  * <p> The result of any attempt to modify a node passed to a
64  * <code>LSSerializerFilter</code> is implementation dependent.
65  * <p> DOM applications must not raise exceptions in a filter. The effect of
66  * throwing exceptions from a filter is DOM implementation dependent.
67  * <p> For efficiency, a node passed to the filter may not be the same as the
68  * one that is actually in the tree. And the actual node (node object
69  * identity) may be reused during the process of filtering and serializing a
70  * document.
71  * <p> See also the http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407 Document Object
72  * Model (DOM) Level 3 Load
73  * and Save Specification</a>.
74  */
75
76 public interface LSSerializerFilter extends NodeFilter {
77     /**
78      * Tells the <code>LSSerializer</code> what types of nodes to show to the
79      * filter. If a node is not shown to the filter using this attribute, it

```

```

78      * is automatically serialized. See <code>NodeFilter</code> for
79      * definition of the constants. The constants <code>SHOW_DOCUMENT</code>
80      * , <code>SHOW_DOCUMENT_TYPE</code>, <code>SHOW_DOCUMENT_FRAGMENT</code>
81      * , <code>SHOW_NOTATION</code>, and <code>SHOW_ENTITY</code> are
82      * meaningless here, such nodes will never be passed to a
83      * <code>LSSerializerFilter</code>.
84      * <br> Unlike [<code>SHOW\_ATTRIBUTE</code> constant indicates that the
86      \* <code>Attr</code> nodes are shown and passed to the filter.
87      \* <br> The constants used here are defined in \[

```

## 42 Dom Ranges (org.w3c.dom.ranges package)

### 42.1 DocumentRange.java

Secuencia 458: org.w3c.dom.ranges.DocumentRange.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *

```

```

31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.ranges;
43
44  /**
45   * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113'>
      Document Object Model (DOM) Level 2 Traversal and Range Specification</a>.
46   * @since DOM Level 2
47   */
48  public interface DocumentRange {
49      /**
50       * This interface can be obtained from the object implementing the
51       * <code>Document</code> interface using binding-specific casting
52       * methods.
53       * @return The initial state of the Range returned from this method is
54       * such that both of its boundary-points are positioned at the
55       * beginning of the corresponding Document, before any content. The
56       * Range returned can only be used to select content associated with
57       * this Document, or with DocumentFragments and Attrs for which this
58       * Document is the <code>ownerDocument</code>.
59       */
60      public Range createRange();
61
62  }

```

## 42.2 Range.java

Secuencia 459: org/w3c/dom/ranges/Range.java

```

1  /*
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *

```

```
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.ranges;
43
44  import org.w3c.dom.Node;
45  import org.w3c.dom.DOMException;
46  import org.w3c.dom.DocumentFragment;
47
48  /**
49   * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113'>
      Document Object Model (DOM) Level 2 Traversal and Range Specification</a>.
50   * @since DOM Level 2
51   */
52  public interface Range {
53      /**
54       * Node within which the Range begins
55       * @exception DOMException
56       *     INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
57       *     invoked on this object.
58       */
59      public Node getStartContainer()
60              throws DOMException;
61
62      /**
63       * Offset within the starting node of the Range.
64       * @exception DOMException
65       *     INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
66       *     invoked on this object.
67       */
68      public int getStartOffset()
```



```

69         throws DOMException;
70
71     /**
72     * Node within which the Range ends
73     * @exception DOMException
74     *     INVALID_STATE_ERR: Raised if detach() has already been
75     *     invoked on this object.
76     */
77     public Node getEndContainer()
78         throws DOMException;
79
80     /**
81     * Offset within the ending node of the Range.
82     * @exception DOMException
83     *     INVALID_STATE_ERR: Raised if detach() has already been
84     *     invoked on this object.
85     */
86     public int getEndOffset()
87         throws DOMException;
88
89     /**
90     * TRUE if the Range is collapsed
91     * @exception DOMException
92     *     INVALID_STATE_ERR: Raised if detach() has already been
93     *     invoked on this object.
94     */
95     public boolean getCollapsed()
96         throws DOMException;
97
98     /**
99     * The deepest common ancestor container of the Range's two
100    * boundary-points.
101    * @exception DOMException
102    *     INVALID_STATE_ERR: Raised if detach() has already been
103    *     invoked on this object.
104    */
105    public Node getCommonAncestorContainer()
106        throws DOMException;
107
108    /**
109    * Sets the attributes describing the start of the Range.
110    * @param refNode The refNode value. This parameter must be
111    *     different from null.
112    * @param offset The startOffset value.
113    * @exception RangeException
114    *     INVALID_NODE_TYPE_ERR: Raised if refNode or an ancestor
115    *     of refNode is an Entity, Notation, or DocumentType
116    *     node.
117    * @exception DOMException
118    *     INDEX_SIZE_ERR: Raised if offset is negative or greater
119    *     than the number of child units in refNode. Child units
120    *     are 16-bit units if refNode is a type of CharacterData
121    *     node (e.g., a Text or Comment node) or a ProcessingInstruction

```

```

122     * node. Child units are Nodes in all other cases.
123     * <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
124     * been invoked on this object.
125     * <br>WRONG_DOCUMENT_ERR: Raised if <code>refNode</code> was created
126     * from a different document than the one that created this range.
127     */
128     public void setStart(Node refNode,
129                          int offset)
130         throws RangeException, DOMException;
131
132     /**
133     * Sets the attributes describing the end of a Range.
134     * @param refNode The <code>refNode</code> value. This parameter must be
135     * different from <code>null</code>.
136     * @param offset The <code>endOffset</code> value.
137     * @exception RangeException
138     *     INVALID_NODE_TYPE_ERR: Raised if <code>refNode</code> or an ancestor
139     * of <code>refNode</code> is an Entity, Notation, or DocumentType
140     * node.
141     * @exception DOMException
142     *     INDEX_SIZE_ERR: Raised if <code>offset</code> is negative or greater
143     * than the number of child units in <code>refNode</code>. Child units
144     * are 16-bit units if <code>refNode</code> is a type of CharacterData
145     * node (e.g., a Text or Comment node) or a ProcessingInstruction
146     * node. Child units are Nodes in all other cases.
147     * <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
148     * been invoked on this object.
149     * <br>WRONG_DOCUMENT_ERR: Raised if <code>refNode</code> was created
150     * from a different document than the one that created this range.
151     */
152     public void setEnd(Node refNode,
153                       int offset)
154         throws RangeException, DOMException;
155
156     /**
157     * Sets the start position to be before a node
158     * @param refNode Range starts before <code>refNode</code>
159     * @exception RangeException
160     *     INVALID_NODE_TYPE_ERR: Raised if the root container of
161     * <code>refNode</code> is not an Attr, Document, or DocumentFragment
162     * node or if <code>refNode</code> is a Document, DocumentFragment,
163     * Attr, Entity, or Notation node.
164     * @exception DOMException
165     *     INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
166     * invoked on this object.
167     * <br>WRONG_DOCUMENT_ERR: Raised if <code>refNode</code> was created
168     * from a different document than the one that created this range.
169     */
170     public void setStartBefore(Node refNode)
171         throws RangeException, DOMException;
172
173     /**
174     * Sets the start position to be after a node

```

```

175     * @param refNode Range starts after <code>refNode</code>
176     * @exception RangeException
177     *     INVALID_NODE_TYPE_ERR: Raised if the root container of
178     *     <code>refNode</code> is not an Attr, Document, or DocumentFragment
179     *     node or if <code>refNode</code> is a Document, DocumentFragment,
180     *     Attr, Entity, or Notation node.
181     * @exception DOMException
182     *     INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
183     *     invoked on this object.
184     *     <br>WRONG_DOCUMENT_ERR: Raised if <code>refNode</code> was created
185     *     from a different document than the one that created this range.
186     */
187     public void setStartAfter(Node refNode)
188         throws RangeException, DOMException;
189
190     /**
191     * Sets the end position to be before a node.
192     * @param refNode Range ends before <code>refNode</code>
193     * @exception RangeException
194     *     INVALID_NODE_TYPE_ERR: Raised if the root container of
195     *     <code>refNode</code> is not an Attr, Document, or DocumentFragment
196     *     node or if <code>refNode</code> is a Document, DocumentFragment,
197     *     Attr, Entity, or Notation node.
198     * @exception DOMException
199     *     INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
200     *     invoked on this object.
201     *     <br>WRONG_DOCUMENT_ERR: Raised if <code>refNode</code> was created
202     *     from a different document than the one that created this range.
203     */
204     public void setEndBefore(Node refNode)
205         throws RangeException, DOMException;
206
207     /**
208     * Sets the end of a Range to be after a node
209     * @param refNode Range ends after <code>refNode</code>.
210     * @exception RangeException
211     *     INVALID_NODE_TYPE_ERR: Raised if the root container of
212     *     <code>refNode</code> is not an Attr, Document or DocumentFragment
213     *     node or if <code>refNode</code> is a Document, DocumentFragment,
214     *     Attr, Entity, or Notation node.
215     * @exception DOMException
216     *     INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
217     *     invoked on this object.
218     *     <br>WRONG_DOCUMENT_ERR: Raised if <code>refNode</code> was created
219     *     from a different document than the one that created this range.
220     */
221     public void setEndAfter(Node refNode)
222         throws RangeException, DOMException;
223
224     /**
225     * Collapse a Range onto one of its boundary-points
226     * @param toStart If TRUE, collapses the Range onto its start; if FALSE,
227     *     collapses it onto its end.

```

```

228     * @exception DOMException
229     *     INVALID_STATE_ERR: Raised if detach() has already been
230     *     invoked on this object.
231     */
232     public void collapse(boolean toStart)
233         throws DOMException;
234
235     /**
236     * Select a node and its contents
237     * @param refNode The node to select.
238     * @exception RangeException
239     *     INVALID_NODE_TYPE_ERR: Raised if an ancestor of refNode
240     *     is an Entity, Notation or DocumentType node or if
241     *     refNode is a Document, DocumentFragment, Attr, Entity,
242     *     or Notation node.
243     * @exception DOMException
244     *     INVALID_STATE_ERR: Raised if detach() has already been
245     *     invoked on this object.
246     *     <br>WRONG_DOCUMENT_ERR: Raised if refNode was created
247     *     from a different document than the one that created this range.
248     */
249     public void selectNode(Node refNode)
250         throws RangeException, DOMException;
251
252     /**
253     * Select the contents within a node
254     * @param refNode Node to select from
255     * @exception RangeException
256     *     INVALID_NODE_TYPE_ERR: Raised if refNode or an ancestor
257     *     of refNode is an Entity, Notation or DocumentType node.
258     * @exception DOMException
259     *     INVALID_STATE_ERR: Raised if detach() has already been
260     *     invoked on this object.
261     *     <br>WRONG_DOCUMENT_ERR: Raised if refNode was created
262     *     from a different document than the one that created this range.
263     */
264     public void selectNodeContents(Node refNode)
265         throws RangeException, DOMException;
266
267     // CompareHow
268     /**
269     * Compare start boundary-point of sourceRange to start
270     * boundary-point of Range on which compareBoundaryPoints
271     * is invoked.
272     */
273     public static final short START_TO_START = 0;
274     /**
275     * Compare start boundary-point of sourceRange to end
276     * boundary-point of Range on which compareBoundaryPoints
277     * is invoked.
278     */
279     public static final short START_TO_END = 1;
280     /**

```

```

281     * Compare end boundary-point of <code>sourceRange</code> to end
282     * boundary-point of Range on which <code>compareBoundaryPoints</code>
283     * is invoked.
284     */
285     public static final short END_TO_END                = 2;
286     /**
287     * Compare end boundary-point of <code>sourceRange</code> to start
288     * boundary-point of Range on which <code>compareBoundaryPoints</code>
289     * is invoked.
290     */
291     public static final short END_TO_START              = 3;
292
293     /**
294     * Compare the boundary-points of two Ranges in a document.
295     * @param how A code representing the type of comparison, as defined
296     *   above.
297     * @param sourceRange The <code>Range</code> on which this current
298     *   <code>Range</code> is compared to.
299     * @return -1, 0 or 1 depending on whether the corresponding
300     *   boundary-point of the Range is respectively before, equal to, or
301     *   after the corresponding boundary-point of <code>sourceRange</code>.
302     * @exception DOMException
303     *   WRONG_DOCUMENT_ERR: Raised if the two Ranges are not in the same
304     *   Document or DocumentFragment.
305     *   <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
306     *   been invoked on this object.
307     */
308     public short compareBoundaryPoints(short how,
309                                       Range sourceRange)
310         throws DOMException;
311
312     /**
313     * Removes the contents of a Range from the containing document or
314     * document fragment without returning a reference to the removed
315     * content.
316     * @exception DOMException
317     *   NO_MODIFICATION_ALLOWED_ERR: Raised if any portion of the content of
318     *   the Range is read-only or any of the nodes that contain any of the
319     *   content of the Range are read-only.
320     *   <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
321     *   been invoked on this object.
322     */
323     public void deleteContents()
324         throws DOMException;
325
326     /**
327     * Moves the contents of a Range from the containing document or document
328     * fragment to a new DocumentFragment.
329     * @return A DocumentFragment containing the extracted contents.
330     * @exception DOMException
331     *   NO_MODIFICATION_ALLOWED_ERR: Raised if any portion of the content of
332     *   the Range is read-only or any of the nodes which contain any of the
333     *   content of the Range are read-only.

```

```

334 * <br>HIERARCHY_REQUEST_ERR: Raised if a DocumentType node would be
335 * extracted into the new DocumentFragment.
336 * <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
337 * been invoked on this object.
338 */
339 public DocumentFragment extractContents()
340     throws DOMException;
341
342 /**
343 * Duplicates the contents of a Range
344 * @return A DocumentFragment that contains content equivalent to this
345 * Range.
346 * @exception DOMException
347 * HIERARCHY_REQUEST_ERR: Raised if a DocumentType node would be
348 * extracted into the new DocumentFragment.
349 * <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
350 * been invoked on this object.
351 */
352 public DocumentFragment cloneContents()
353     throws DOMException;
354
355 /**
356 * Inserts a node into the Document or DocumentFragment at the start of
357 * the Range. If the container is a Text node, this will be split at the
358 * start of the Range (as if the Text node's splitText method was
359 * performed at the insertion point) and the insertion will occur
360 * between the two resulting Text nodes. Adjacent Text nodes will not be
361 * automatically merged. If the node to be inserted is a
362 * DocumentFragment node, the children will be inserted rather than the
363 * DocumentFragment node itself.
364 * @param newNode The node to insert at the start of the Range
365 * @exception DOMException
366 * NO_MODIFICATION_ALLOWED_ERR: Raised if an ancestor container of the
367 * start of the Range is read-only.
368 * <br>WRONG_DOCUMENT_ERR: Raised if <code>newNode</code> and the
369 * container of the start of the Range were not created from the same
370 * document.
371 * <br>HIERARCHY_REQUEST_ERR: Raised if the container of the start of
372 * the Range is of a type that does not allow children of the type of
373 * <code>newNode</code> or if <code>newNode</code> is an ancestor of
374 * the container.
375 * <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
376 * been invoked on this object.
377 * @exception RangeException
378 * INVALID_NODE_TYPE_ERR: Raised if <code>newNode</code> is an Attr,
379 * Entity, Notation, or Document node.
380 */
381 public void insertNode(Node newNode)
382     throws DOMException, RangeException;
383
384 /**
385 * Reparents the contents of the Range to the given node and inserts the
386 * node at the position of the start of the Range.

```

```

387 * @param newParent The node to surround the contents with.
388 * @exception DOMException
389 *   NO_MODIFICATION_ALLOWED_ERR: Raised if an ancestor container of
390 *   either boundary-point of the Range is read-only.
391 *   <br>WRONG_DOCUMENT_ERR: Raised if <code> newParent</code> and the
392 *   container of the start of the Range were not created from the same
393 *   document.
394 *   <br>HIERARCHY_REQUEST_ERR: Raised if the container of the start of
395 *   the Range is of a type that does not allow children of the type of
396 *   <code>newParent</code> or if <code>newParent</code> is an ancestor
397 *   of the container or if <code>node</code> would end up with a child
398 *   node of a type not allowed by the type of <code>node</code>.
399 *   <br>INVALID_STATE_ERR: Raised if <code>detach()</code> has already
400 *   been invoked on this object.
401 * @exception RangeException
402 *   BAD_BOUNDARYPOINTS_ERR: Raised if the Range partially selects a
403 *   non-text node.
404 *   <br>INVALID_NODE_TYPE_ERR: Raised if <code> node</code> is an Attr,
405 *   Entity, DocumentType, Notation, Document, or DocumentFragment node.
406 */
407 public void surroundContents(Node newParent)
408     throws DOMException, RangeException;
409
410 /**
411 * Produces a new Range whose boundary-points are equal to the
412 * boundary-points of the Range.
413 * @return The duplicated Range.
414 * @exception DOMException
415 *   INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
416 *   invoked on this object.
417 */
418 public Range cloneRange()
419     throws DOMException;
420
421 /**
422 * Returns the contents of a Range as a string. This string contains only
423 * the data characters, not any markup.
424 * @return The contents of the Range.
425 * @exception DOMException
426 *   INVALID_STATE_ERR: Raised if <code>detach()</code> has already been
427 *   invoked on this object.
428 */
429 public String toString()
430     throws DOMException;
431
432 /**
433 * Called to indicate that the Range is no longer in use and that the
434 * implementation may relinquish any resources associated with this
435 * Range. Subsequent calls to any methods or attribute getters on this
436 * Range will result in a <code>DOMException</code> being thrown with an
437 * error code of <code>INVALID_STATE_ERR</code>.
438 * @exception DOMException
439 *   INVALID_STATE_ERR: Raised if <code>detach()</code> has already been

```

```
440     *   invoked on this object.
441     */
442     public void detach()
443         throws DOMException;
444
445 }
```

### 42.3 RangeException.java

Secuencia 460: org/w3c/dom/ranges/RangeException.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.ranges;
43
```



```

44  /**
45   * Range operations may throw a RangeException as specified in
46   * their method descriptions.
47   * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113'>
      Document Object Model (DOM) Level 2 Traversal and Range Specification</a>.
48   * @since DOM Level 2
49   */
50  public class RangeException extends RuntimeException {
51      public RangeException(short code, String message) {
52          super(message);
53          this.code = code;
54      }
55      public short    code;
56      // RangeExceptionCode
57      /**
58       * If the boundary-points of a Range do not meet specific requirements.
59       */
60      public static final short BAD_BOUNDARYPOINTS_ERR    = 1;
61      /**
62       * If the container of an boundary-point of a Range is being set to either
63       * a node of an invalid type or a node with an ancestor of an invalid
64       * type.
65       */
66      public static final short INVALID_NODE_TYPE_ERR    = 2;
67  }
68  }

```

## 43 Dom Stylesheets (org.w3c.dom.stylesheets package)

### 43.1 DocumentStyle.java

Secuencia 461: org.w3c.dom.stylesheets/DocumentStyle.java

```

1  /**
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *

```

```

22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.stylesheets;
43
44  /**
45   * The DocumentStyle interface provides a mechanism by which the
46   * style sheets embedded in a document can be retrieved. The expectation is
47   * that an instance of the DocumentStyle interface can be
48   * obtained by using binding-specific casting methods on an instance of the
49   * Document interface.
50   * 

See also the Document
51   * Object Model (DOM) Level 2 Style Specification.


52   * @since DOM Level 2
53   */
54  public interface DocumentStyle {
55      /**
56       * A list containing all the style sheets explicitly linked into or
57       * embedded in a document. For HTML documents, this includes external
58       * style sheets, included via the HTML LINK element, and inline STYLE
59       * elements. In XML, this includes external style sheets, included via
60       * style sheet processing instructions (see [XML StyleSheet]).
61       */
62      public StyleSheetList getStyleSheets();
63  }

```

## 43.2 LinkStyle.java

Secuencia 462: org/w3c/dom/stylesheets/LinkStyle.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *

```

```

7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.stylesheets;
43
44 /**
45  * The LinkStyle interface provides a mechanism by which a style
46  * sheet can be retrieved from the node responsible for linking it into a
47  * document. An instance of the LinkStyle interface can be
48  * obtained using binding-specific casting methods on an instance of a
49  * linking node (HTMLLinkElement, HTMLStyleElement
50  * or ProcessingInstruction in DOM Level 2).
51  * 

See also the Document Object Model \(DOM\) Level 2 Style Specification.


52  * @since DOM Level 2
53  */
54 public interface LinkStyle {
55     /**
56      * The style sheet.
57      */
58     public StyleSheet getSheet();

```

```
59  
60 }
```

### 43.3 MediaList.java

Secuencia 463: org/w3c/dom/stylesheets/MediaList.java

```
1  /*  
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.  
3  *  
4  *  
5  *  
6  *  
7  *  
8  *  
9  *  
10 *  
11 *  
12 *  
13 *  
14 *  
15 *  
16 *  
17 *  
18 *  
19 *  
20 *  
21 *  
22 *  
23 */  
24  
25 /*  
26 *  
27 *  
28 *  
29 *  
30 *  
31 * Copyright (c) 2000 World Wide Web Consortium,  
32 * (Massachusetts Institute of Technology, Institut National de  
33 * Recherche en Informatique et en Automatique, Keio University). All  
34 * Rights Reserved. This program is distributed under the W3C's Software  
35 * Intellectual Property License. This program is distributed in the  
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even  
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR  
38 * PURPOSE.  
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.  
40 */  
41  
42 package org.w3c.dom.stylesheets;  
43  
44 import org.w3c.dom.DOMException;  
45  
46 /**  
47 * The MediaList interface provides the abstraction of an
```

```

48 * ordered collection of media, without defining or constraining how this
49 * collection is implemented. An empty list is the same as a list that
50 * contains the medium <code>"all"</code>.
51 * <p> The items in the <code>MediaList</code> are accessible via an integral
52 * index, starting from 0.
53 * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
    Object Model (DOM) Level 2 Style Specification</a>.
54 * @since DOM Level 2
55 */
56 public interface MediaList {
57     /**
58      * The parsable textual representation of the media list. This is a
59      * comma-separated list of media.
60      */
61     public String getMediaText();
62     /**
63      * The parsable textual representation of the media list. This is a
64      * comma-separated list of media.
65      * @exception DOMException
66      *     SYNTAX_ERR: Raised if the specified string value has a syntax error
67      *     and is unparseable.
68      *     <br>NO_MODIFICATION_ALLOWED_ERR: Raised if this media list is
69      *     readonly.
70      */
71     public void setMediaText(String mediaText)
72                             throws DOMException;
73
74     /**
75      * The number of media in the list. The range of valid media is
76      * <code>0</code> to <code>length-1</code> inclusive.
77      */
78     public int getLength();
79
80     /**
81      * Returns the <code>index</code>th in the list. If <code>index</code> is
82      * greater than or equal to the number of media in the list, this
83      * returns <code>null</code>.
84      * @param index    Index into the collection.
85      * @return The medium at the <code>index</code>th position in the
86      *     <code>MediaList</code>, or <code>null</code> if that is not a valid
87      *     index.
88      */
89     public String item(int index);
90
91     /**
92      * Deletes the medium indicated by <code>oldMedium</code> from the list.
93      * @param oldMedium The medium to delete in the media list.
94      * @exception DOMException
95      *     NO_MODIFICATION_ALLOWED_ERR: Raised if this list is readonly.
96      *     <br> NOT_FOUND_ERR: Raised if <code>oldMedium</code> is not in the
97      *     list.
98      */
99     public void deleteMedium(String oldMedium)

```

```

100         throws DOMException;
101
102     /**
103     * Adds the medium <code>newMedium</code> to the end of the list. If the
104     * <code>newMedium</code> is already used, it is first removed.
105     * @param newMedium The new medium to add.
106     * @exception DOMException
107     *     INVALID_CHARACTER_ERR: If the medium contains characters that are
108     *     invalid in the underlying style language.
109     *     <br> NO_MODIFICATION_ALLOWED_ERR: Raised if this list is readonly.
110     */
111     public void appendMedium(String newMedium)
112         throws DOMException;
113
114 }

```

#### 43.4 StyleSheet.java

Secuencia 464: [org.w3c/dom/stylesheets/StyleSheet.java](http://org.w3c/dom/stylesheets/StyleSheet.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software

```

```

35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.stylesheets;
43
44  import org.w3c.dom.Node;
45
46  /**
47   * The StyleSheet interface is the abstract base interface for
48   * any type of style sheet. It represents a single style sheet associated
49   * with a structured document. In HTML, the StyleSheet interface represents
50   * either an external style sheet, included via the HTML LINK element, or
51   * an inline STYLE element. In XML, this interface represents an external
52   * style sheet, included via a style sheet processing instruction.
53   * 

See also the Document
54   * Object Model (DOM) Level 2 Style Specification.


55   * @since DOM Level 2
56   */
57  public interface StyleSheet {
58      /**
59       * This specifies the style sheet language for this style sheet. The
60       * style sheet language is specified as a content type (e.g.
61       * "text/css"). The content type is often specified in the
62       * ownerNode. Also see the type attribute definition for
63       * the LINK element in HTML 4.0, and the type
64       * pseudo-attribute for the XML style sheet processing instruction.
65       */
66      public String getType();
67
68      /**
69       * false if the style sheet is applied to the document.
70       * true if it is not. Modifying this attribute may cause a
71       * new resolution of style for the document. A stylesheet only applies
72       * if both an appropriate medium definition is present and the disabled
73       * attribute is false. So, if the media doesn't apply to the current
74       * user agent, the disabled attribute is ignored.
75       */
76      public boolean getDisabled();
77
78      /**
79       * false if the style sheet is applied to the document.
80       * true if it is not. Modifying this attribute may cause a
81       * new resolution of style for the document. A stylesheet only applies
82       * if both an appropriate medium definition is present and the disabled
83       * attribute is false. So, if the media doesn't apply to the current
84       * user agent, the disabled attribute is ignored.
85       */
86      public void setDisabled(boolean disabled);
87
88      /**

```

```

87      * The node that associates this style sheet with the document. For HTML,
88      * this may be the corresponding <code>LINK</code> or <code>STYLE</code>
89      * element. For XML, it may be the linking processing instruction. For
90      * style sheets that are included by other style sheets, the value of
91      * this attribute is <code>null</code>.
92      */
93      public Node getOwnerNode();
94
95      /**
96      * For style sheet languages that support the concept of style sheet
97      * inclusion, this attribute represents the including style sheet, if
98      * one exists. If the style sheet is a top-level style sheet, or the
99      * style sheet language does not support inclusion, the value of this
100     * attribute is <code>null</code>.
101     */
102     public StyleSheet getParentStyleSheet();
103
104     /**
105     * If the style sheet is a linked style sheet, the value of its attribute
106     * is its location. For inline style sheets, the value of this attribute
107     * is <code>null</code>. See the href attribute definition for the
108     * <code>LINK</code> element in HTML 4.0, and the href pseudo-attribute
109     * for the XML style sheet processing instruction.
110     */
111     public String getHref();
112
113     /**
114     * The advisory title. The title is often specified in the
115     * <code>ownerNode</code>. See the title attribute definition for the
116     * <code>LINK</code> element in HTML 4.0, and the title pseudo-attribute
117     * for the XML style sheet processing instruction.
118     */
119     public String getTitle();
120
121     /**
122     * The intended destination media for style information. The media is
123     * often specified in the <code>ownerNode</code>. If no media has been
124     * specified, the <code>MediaList</code> will be empty. See the media
125     * attribute definition for the <code>LINK</code> element in HTML 4.0,
126     * and the media pseudo-attribute for the XML style sheet processing
127     * instruction . Modifying the media list may cause a change to the
128     * attribute <code>disabled</code>.
129     */
130     public MediaList getMedia();
131
132 }

```

## 43.5 StyleSheetList.java

Secuencia 465: [org.w3c/dom/stylesheets/StyleSheetList.java](http://org.w3c/dom/stylesheets/StyleSheetList.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *

```



```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.stylesheets;
43
44 /**
45  * The StyleSheetList interface provides the abstraction of an
46  * ordered collection of style sheets.
47  * <p> The items in the StyleSheetList are accessible via an
48  * integral index, starting from 0.
49  * <p> See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113'>Document
    Object Model (DOM) Level 2 Style Specification</a>.
50  * @since DOM Level 2
51  */
52 public interface StyleSheetList {
53     /**
54      * The number of StyleSheets in the list. The range of valid
55      * child stylesheet indices is 0 to length-1
```

```

56     * inclusive.
57     */
58     public int getLength();
59
60     /**
61     * Used to retrieve a style sheet by ordinal index. If index is greater
62     * than or equal to the number of style sheets in the list, this returns
63     * <code>null</code>.
64     * @param index Index into the collection
65     * @return The style sheet at the <code>index</code> position in the
66     *         <code>StyleSheetList</code>, or <code>null</code> if that is not a
67     *         valid index.
68     */
69     public StyleSheet item(int index);
70
71 }

```

## 44 Dom Traversal (org.w3c.dom.traversal package)

### 44.1 DocumentTraversal.java

Secuencia 466: org/w3c/dom/traversal/DocumentTraversal.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,

```



```

84         NodeFilter filter,
85         boolean entityReferenceExpansion)
86         throws DOMException;
87
88     /**
89     * Create a new TreeWalker over the subtree rooted at the
90     * specified node.
91     * @param root The node which will serve as the root for the
92     * TreeWalker. The whatToShow flags and the
93     * NodeFilter are not considered when setting this value;
94     * any node type will be accepted as the root. The
95     * currentNode of the TreeWalker is
96     * initialized to this node, whether or not it is visible. The
97     * root functions as a stopping point for traversal
98     * methods that look upward in the document structure, such as
99     * parentNode and nextNode. The root must
100    * not be null.
101    * @param whatToShow This flag specifies which node types may appear in
102    * the logical view of the tree presented by the
103    * TreeWalker. See the description of
104    * NodeFilter for the set of possible SHOW_
105    * values. These flags can be combined using OR.
106    * @param filter The NodeFilter to be used with this
107    * TreeWalker, or null to indicate no filter.
108    * @param entityReferenceExpansion If this flag is false, the contents of
109    * EntityReference nodes are not presented in the logical
110    * view.
111    * @return The newly created TreeWalker.
112    * @exception DOMException
113    *     NOT_SUPPORTED_ERR: Raised if the specified root is
114    *     null.
115    */
116    public TreeWalker createTreeWalker(Node root,
117                                       int whatToShow,
118                                       NodeFilter filter,
119                                       boolean entityReferenceExpansion)
120                                       throws DOMException;
121
122 }

```

## 44.2 NodeFilter.java

Secuencia 467: [org.w3c/dom/traversal/NodeFilter.java](http://org.w3c/dom/traversal/NodeFilter.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```

```
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2000 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.traversal;
43
44  import org.w3c.dom.Node;
45
46  /**
47   * Filters are objects that know how to "filter out" nodes. If a
48   * NodeIterator or TreeWalker is given a
49   * NodeFilter, it applies the filter before it returns the next
50   * node. If the filter says to accept the node, the traversal logic returns
51   * it; otherwise, traversal looks for the next node and pretends that the
52   * node that was rejected was not there.
53   * 

The DOM does not provide any filters. NodeFilter is just an
54   * interface that users can implement to provide their own filters.
55   * 

NodeFilters do not need to know how to traverse from node
56   * to node, nor do they need to know anything about the data structure that
57   * is being traversed. This makes it very easy to write filters, since the
58   * only thing they have to know how to do is evaluate a single node. One
59   * filter may be used with a number of different kinds of traversals,
60   * encouraging code reuse.
61   * 

See also the http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113
62   * Document Object Model (DOM) Level 2 Traversal and Range Specification.


63   * @since DOM Level 2


```

```

63  */
64  public interface NodeFilter {
65      // Constants returned by acceptNode
66      /**
67       * Accept the node. Navigation methods defined for
68       * <code>NodeIterator</code> or <code>TreeWalker</code> will return this
69       * node.
70       */
71      public static final short FILTER_ACCEPT          = 1;
72      /**
73       * Reject the node. Navigation methods defined for
74       * <code>NodeIterator</code> or <code>TreeWalker</code> will not return
75       * this node. For <code>TreeWalker</code>, the children of this node
76       * will also be rejected. <code>NodeIterators</code> treat this as a
77       * synonym for <code>FILTER_SKIP</code>.
78       */
79      public static final short FILTER_REJECT          = 2;
80      /**
81       * Skip this single node. Navigation methods defined for
82       * <code>NodeIterator</code> or <code>TreeWalker</code> will not return
83       * this node. For both <code>NodeIterator</code> and
84       * <code>TreeWalker</code>, the children of this node will still be
85       * considered.
86       */
87      public static final short FILTER_SKIP            = 3;
88
89      // Constants for whatToShow
90      /**
91       * Show all <code>Nodes</code>.
92       */
93      public static final int SHOW_ALL                  = 0xFFFFFFFF;
94      /**
95       * Show <code>Element</code> nodes.
96       */
97      public static final int SHOW_ELEMENT              = 0x00000001;
98      /**
99       * Show <code>Attr</code> nodes. This is meaningful only when creating an
100      * <code>NodeIterator</code> or <code>TreeWalker</code> with an
101      * attribute node as its <code>root</code>; in this case, it means that
102      * the attribute node will appear in the first position of the iteration
103      * or traversal. Since attributes are never children of other nodes,
104      * they do not appear when traversing over the document tree.
105      */
106      public static final int SHOW_ATTRIBUTE            = 0x00000002;
107      /**
108       * Show <code>Text</code> nodes.
109       */
110      public static final int SHOW_TEXT                  = 0x00000004;
111      /**
112       * Show <code>CDATASection</code> nodes.
113       */
114      public static final int SHOW_CDATA_SECTION        = 0x00000008;
115      /**

```

```
116     * Show EntityReference nodes.
117     */
118     public static final int SHOW_ENTITY_REFERENCE      = 0x00000010;
119     /**
120     * Show Entity nodes. This is meaningful only when creating
121     * an NodeIterator or TreeWalker with an
122     * Entity node as its root; in this case, it
123     * means that the Entity node will appear in the first
124     * position of the traversal. Since entities are not part of the
125     * document tree, they do not appear when traversing over the document
126     * tree.
127     */
128     public static final int SHOW_ENTITY                = 0x00000020;
129     /**
130     * Show ProcessingInstruction nodes.
131     */
132     public static final int SHOW_PROCESSING_INSTRUCTION = 0x00000040;
133     /**
134     * Show Comment nodes.
135     */
136     public static final int SHOW_COMMENT               = 0x00000080;
137     /**
138     * Show Document nodes.
139     */
140     public static final int SHOW_DOCUMENT             = 0x00000100;
141     /**
142     * Show DocumentType nodes.
143     */
144     public static final int SHOW_DOCUMENT_TYPE        = 0x00000200;
145     /**
146     * Show DocumentFragment nodes.
147     */
148     public static final int SHOW_DOCUMENT_FRAGMENT    = 0x00000400;
149     /**
150     * Show Notation nodes. This is meaningful only when creating
151     * an NodeIterator or TreeWalker with a
152     * Notation node as its root; in this case, it
153     * means that the Notation node will appear in the first
154     * position of the traversal. Since notations are not part of the
155     * document tree, they do not appear when traversing over the document
156     * tree.
157     */
158     public static final int SHOW_NOTATION             = 0x00000800;
159
160     /**
161     * Test whether a specified node is visible in the logical view of a
162     * TreeWalker or NodeIterator. This function
163     * will be called by the implementation of TreeWalker and
164     * NodeIterator; it is not normally called directly from
165     * user code. (Though you could do so if you wanted to use the same
166     * filter to guide your own application logic.)
167     * @param n The node to check to see if it passes the filter or not.
168     * @return A constant to determine whether the node is accepted,
```

```
169     *   rejected, or skipped, as defined above.
170     */
171     public short acceptNode(Node n);
172
173 }
```

### 44.3 NodeIterator.java

Secuencia 468: org/w3c/dom/traversal/NodeIterator.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.traversal;
43
44 import org.w3c.dom.Node;
```



```

45 import org.w3c.dom.DOMException;
46
47 /**
48  * <code>NodeIterators</code> are used to step through a set of nodes, e.g.
49  * the set of nodes in a <code>NodeList</code>, the document subtree
50  * governed by a particular <code>Node</code>, the results of a query, or
51  * any other set of nodes. The set of nodes to be iterated is determined by
52  * the implementation of the <code>NodeIterator</code>. DOM Level 2
53  * specifies a single <code>NodeIterator</code> implementation for
54  * document-order traversal of a document subtree. Instances of these
55  * <code>NodeIterators</code> are created by calling
56  * <code>DocumentTraversal</code>.<code>.createNodeIterator()</code>.
57  * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113'>
    Document Object Model (DOM) Level 2 Traversal and Range Specification</a>.
58  * @since DOM Level 2
59  */
60 public interface NodeIterator {
61     /**
62      * The root node of the <code>NodeIterator</code>, as specified when it
63      * was created.
64      */
65     public Node getRoot();
66
67     /**
68      * This attribute determines which node types are presented via the
69      * <code>NodeIterator</code>. The available set of constants is defined
70      * in the <code>NodeFilter</code> interface. Nodes not accepted by
71      * <code>whatToShow</code> will be skipped, but their children may still
72      * be considered. Note that this skip takes precedence over the filter,
73      * if any.
74      */
75     public int getWhatToShow();
76
77     /**
78      * The <code>NodeFilter</code> used to screen nodes.
79      */
80     public NodeFilter getFilter();
81
82     /**
83      * The value of this flag determines whether the children of entity
84      * reference nodes are visible to the <code>NodeIterator</code>. If
85      * false, these children and their descendants will be rejected. Note
86      * that this rejection takes precedence over <code>whatToShow</code> and
87      * the filter. Also note that this is currently the only situation where
88      * <code>NodeIterators</code> may reject a complete subtree rather than
89      * skipping individual nodes.
90      * <br>
91      * <br> To produce a view of the document that has entity references
92      * expanded and does not expose the entity reference node itself, use
93      * the <code>whatToShow</code> flags to hide the entity reference node
94      * and set <code>expandEntityReferences</code> to true when creating the
95      * <code>NodeIterator</code>. To produce a view of the document that has
96      * entity reference nodes but no entity expansion, use the

```

```

97      * <code>whatToShow</code> flags to show the entity reference node and
98      * set <code>expandEntityReferences</code> to false.
99      */
100     public boolean getExpandEntityReferences();
101
102     /**
103      * Returns the next node in the set and advances the position of the
104      * <code>NodeIterator</code> in the set. After a
105      * <code>NodeIterator</code> is created, the first call to
106      * <code>nextNode()</code> returns the first node in the set.
107      * @return The next <code>Node</code> in the set being iterated over, or
108      *         <code>null</code> if there are no more members in that set.
109      * @exception DOMException
110      *         INVALID_STATE_ERR: Raised if this method is called after the
111      *         <code>detach</code> method was invoked.
112      */
113     public Node nextNode()
114         throws DOMException;
115
116     /**
117      * Returns the previous node in the set and moves the position of the
118      * <code>NodeIterator</code> backwards in the set.
119      * @return The previous <code>Node</code> in the set being iterated over,
120      *         or <code>null</code> if there are no more members in that set.
121      * @exception DOMException
122      *         INVALID_STATE_ERR: Raised if this method is called after the
123      *         <code>detach</code> method was invoked.
124      */
125     public Node previousNode()
126         throws DOMException;
127
128     /**
129      * Detaches the <code>NodeIterator</code> from the set which it iterated
130      * over, releasing any computational resources and placing the
131      * <code>NodeIterator</code> in the INVALID state. After
132      * <code>detach</code> has been invoked, calls to <code>nextNode</code>
133      * or <code>previousNode</code> will raise the exception
134      * INVALID_STATE_ERR.
135      */
136     public void detach();
137
138 }

```

#### 44.4 TreeWalker.java

Secuencia 469: [org.w3c/dom/traversal/TreeWalker.java](http://org.w3c/dom/traversal/TreeWalker.java)

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *

```

```
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.traversal;
43
44 import org.w3c.dom.Node;
45 import org.w3c.dom.DOMException;
46
47 /**
48  * <code>TreeWalker</code> objects are used to navigate a document tree or
49  * subtree using the view of the document defined by their
50  * <code>whatToShow</code> flags and filter (if any). Any function which
51  * performs navigation using a <code>TreeWalker</code> will automatically
52  * support any view defined by a <code>TreeWalker</code>.
53  * <p>Omitting nodes from the logical view of a subtree can result in a
54  * structure that is substantially different from the same subtree in the
55  * complete, unfiltered document. Nodes that are siblings in the
56  * <code>TreeWalker</code> view may be children of different, widely
57  * separated nodes in the original view. For instance, consider a
58  * <code>NodeFilter</code> that skips all nodes except for Text nodes and
59  * the root node of a document. In the logical view that results, all text
60  * nodes will be siblings and appear as direct children of the root node, no
```

```

61 * matter how deeply nested the structure of the original document.
62 * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113'>
    Document Object Model (DOM) Level 2 Traversal and Range Specification</a>.
63 * @since DOM Level 2
64 */
65 public interface TreeWalker {
66     /**
67      * The <code>root</code> node of the <code>TreeWalker</code>, as specified
68      * when it was created.
69      */
70     public Node getRoot();
71
72     /**
73      * This attribute determines which node types are presented via the
74      * <code>TreeWalker</code>. The available set of constants is defined in
75      * the <code>NodeFilter</code> interface. Nodes not accepted by
76      * <code>whatToShow</code> will be skipped, but their children may still
77      * be considered. Note that this skip takes precedence over the filter,
78      * if any.
79      */
80     public int getWhatToShow();
81
82     /**
83      * The filter used to screen nodes.
84      */
85     public NodeFilter getFilter();
86
87     /**
88      * The value of this flag determines whether the children of entity
89      * reference nodes are visible to the <code>TreeWalker</code>. If false,
90      * these children and their descendants will be rejected. Note that
91      * this rejection takes precedence over <code>whatToShow</code> and the
92      * filter, if any.
93      * <br> To produce a view of the document that has entity references
94      * expanded and does not expose the entity reference node itself, use
95      * the <code>whatToShow</code> flags to hide the entity reference node
96      * and set <code>expandEntityReferences</code> to true when creating the
97      * <code>TreeWalker</code>. To produce a view of the document that has
98      * entity reference nodes but no entity expansion, use the
99      * <code>whatToShow</code> flags to show the entity reference node and
100     * set <code>expandEntityReferences</code> to false.
101     */
102     public boolean getExpandEntityReferences();
103
104     /**
105      * The node at which the <code>TreeWalker</code> is currently positioned.
106      * <br>Alterations to the DOM tree may cause the current node to no longer
107      * be accepted by the <code>TreeWalker</code>'s associated filter.
108      * <code>currentNode</code> may also be explicitly set to any node,
109      * whether or not it is within the subtree specified by the
110      * <code>root</code> node or would be accepted by the filter and
111      * <code>whatToShow</code> flags. Further traversal occurs relative to
112      * <code>currentNode</code> even if it is not part of the current view,

```

```

113     * by applying the filters in the requested direction; if no traversal
114     * is possible, <code>currentNode</code> is not changed.
115     */
116     public Node getCurrentNode();
117     /**
118     * The node at which the <code>TreeWalker</code> is currently positioned.
119     * <br>Alterations to the DOM tree may cause the current node to no longer
120     * be accepted by the <code>TreeWalker</code>'s associated filter.
121     * <code>currentNode</code> may also be explicitly set to any node,
122     * whether or not it is within the subtree specified by the
123     * <code>root</code> node or would be accepted by the filter and
124     * <code>whatToShow</code> flags. Further traversal occurs relative to
125     * <code>currentNode</code> even if it is not part of the current view,
126     * by applying the filters in the requested direction; if no traversal
127     * is possible, <code>currentNode</code> is not changed.
128     * @exception DOMException
129     *     NOT_SUPPORTED_ERR: Raised if an attempt is made to set
130     *     <code>currentNode</code> to <code>null</code>.
131     */
132     public void setCurrentNode(Node currentNode)
133         throws DOMException;
134
135     /**
136     * Moves to and returns the closest visible ancestor node of the current
137     * node. If the search for <code>parentNode</code> attempts to step
138     * upward from the <code>TreeWalker</code>'s <code>root</code> node, or
139     * if it fails to find a visible ancestor node, this method retains the
140     * current position and returns <code>null</code>.
141     * @return The new parent node, or <code>null</code> if the current node
142     *     has no parent in the <code>TreeWalker</code>'s logical view.
143     */
144     public Node parentNode();
145
146     /**
147     * Moves the <code>TreeWalker</code> to the first visible child of the
148     * current node, and returns the new node. If the current node has no
149     * visible children, returns <code>null</code>, and retains the current
150     * node.
151     * @return The new node, or <code>null</code> if the current node has no
152     *     visible children in the <code>TreeWalker</code>'s logical view.
153     */
154     public Node firstChild();
155
156     /**
157     * Moves the <code>TreeWalker</code> to the last visible child of the
158     * current node, and returns the new node. If the current node has no
159     * visible children, returns <code>null</code>, and retains the current
160     * node.
161     * @return The new node, or <code>null</code> if the current node has no
162     *     children in the <code>TreeWalker</code>'s logical view.
163     */
164     public Node lastChild();
165

```

```

166  /**
167   * Moves the <code>TreeWalker</code> to the previous sibling of the
168   * current node, and returns the new node. If the current node has no
169   * visible previous sibling, returns <code>null</code>, and retains the
170   * current node.
171   * @return The new node, or <code>null</code> if the current node has no
172   * previous sibling. in the <code>TreeWalker</code>'s logical view.
173   */
174  public Node previousSibling();
175
176  /**
177   * Moves the <code>TreeWalker</code> to the next sibling of the current
178   * node, and returns the new node. If the current node has no visible
179   * next sibling, returns <code>null</code>, and retains the current node.
180   * @return The new node, or <code>null</code> if the current node has no
181   * next sibling. in the <code>TreeWalker</code>'s logical view.
182   */
183  public Node nextSibling();
184
185  /**
186   * Moves the <code>TreeWalker</code> to the previous visible node in
187   * document order relative to the current node, and returns the new
188   * node. If the current node has no previous node, or if the search for
189   * <code>previousNode</code> attempts to step upward from the
190   * <code>TreeWalker</code>'s <code>root</code> node, returns
191   * <code>null</code>, and retains the current node.
192   * @return The new node, or <code>null</code> if the current node has no
193   * previous node in the <code>TreeWalker</code>'s logical view.
194   */
195  public Node previousNode();
196
197  /**
198   * Moves the <code>TreeWalker</code> to the next visible node in document
199   * order relative to the current node, and returns the new node. If the
200   * current node has no next node, or if the search for nextNode attempts
201   * to step upward from the <code>TreeWalker</code>'s <code>root</code>
202   * node, returns <code>null</code>, and retains the current node.
203   * @return The new node, or <code>null</code> if the current node has no
204   * next node in the <code>TreeWalker</code>'s logical view.
205   */
206  public Node nextNode();
207
208 }

```

## 45 Dom Views (org.w3c.dom.views package)

### 45.1 AbstractView.java

Secuencia 470: org.w3c/dom/views/AbstractView.java

```

1  /**
2   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3   *
4   *

```

```
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.views;
43
44 /**
45  * A base interface that all views shall derive from.
46  * <p>See also the <a href='http://www.w3.org/TR/2000/REC-DOM-Level-2-Views-20001113'>Document
47  *   Object Model (DOM) Level 2 Views Specification</a>.
48  * @since DOM Level 2
49  */
50 public interface AbstractView {
51     /**
52      * The source <code>DocumentView</code> of which this is an
53      * <code>AbstractView</code>.
54      */
55     public DocumentView getDocument();
56 }
```

## 45.2 DocumentView.java

Secuencia 471: org/w3c/dom/views/DocumentView.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2000 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.views;
43
44 /**
45 * The DocumentView interface is implemented by
46 * Document objects in DOM implementations supporting DOM
47 * Views. It provides an attribute to retrieve the default view of a
48 * document.
49 * 

See also the Document Object Model \(DOM\) Level 2 Views Specification.


```



```

50  * @since DOM Level 2
51  */
52  public interface DocumentView {
53      /**
54       * The default AbstractView for this Document,
55       * or null if none available.
56       */
57      public AbstractView getDefaultView();
58  }
59  }

```

## 46 Dom Xpath (org.w3c.dom.xpath package)

### 46.1 XPathEvaluator.java

Secuencia 472: org.w3c.dom.xpath/XPathEvaluator.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2002 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR

```

[illegible]

```

90                                     throws XPathException, DOMException;
91
92 /**
93  * Adapts any DOM node to resolve namespaces so that an XPath expression
94  * can be easily evaluated relative to the context of the node where it
95  * appeared within the document. This adapter works like the DOM Level 3
96  * method <code>lookupNamespaceURI</code> on nodes in resolving the
97  * namespaceURI from a given prefix using the current information
98  * available in the node's hierarchy at the time lookupNamespaceURI is
99  * called. also correctly resolving the implicit xml prefix.
100  * @param nodeResolver The node to be used as a context for namespace
101  *   resolution.
102  * @return <code>XPathNSResolver</code> which resolves namespaces with
103  *   respect to the definitions in scope for a specified node.
104  */
105 public XPathNSResolver createNSResolver(Node nodeResolver);
106
107 /**
108  * Evaluates an XPath expression string and returns a result of the
109  * specified type if possible.
110  * @param expression The XPath expression string to be parsed and
111  *   evaluated.
112  * @param contextNode The <code>context</code> is context node for the
113  *   evaluation of this XPath expression. If the XPathEvaluator was
114  *   obtained by casting the <code>Document</code> then this must be
115  *   owned by the same document and must be a <code>Document</code>,
116  *   <code>Element</code>, <code>Attribute</code>, <code>Text</code>,
117  *   <code>CDATASection</code>, <code>Comment</code>,
118  *   <code>ProcessingInstruction</code>, or <code>XPathNamespace</code>
119  *   node. If the context node is a <code>Text</code> or a
120  *   <code>CDATASection</code>, then the context is interpreted as the
121  *   whole logical text node as seen by XPath, unless the node is empty
122  *   in which case it may not serve as the XPath context.
123  * @param resolver The <code>resolver</code> permits translation of
124  *   prefixes within the XPath expression into appropriate namespace URIs
125  *   . If this is specified as <code>null</code>, any namespace prefix
126  *   within the expression will result in <code>DOMException</code>
127  *   being thrown with the code <code>NAMESPACE_ERR</code>.
128  * @param type If a specific <code>type</code> is specified, then the
129  *   result will be returned as the corresponding type. For XPath 1.0
130  *   results, this must be one of the codes of the
131  *   <code>XPathResult</code> interface.
132  * @param result The <code>result</code> specifies a specific result
133  *   object which may be reused and returned by this method. If this is
134  *   specified as <code>null</code> or the implementation does not reuse
135  *   the specified result, a new result object will be constructed and
136  *   returned. For XPath 1.0 results, this object will be of type
137  *   <code>XPathResult</code>.
138  * @return The result of the evaluation of the XPath expression. For XPath
139  *   1.0 results, this object will be of type <code>XPathResult</code>.
140  * @exception XPathException
141  *   INVALID_EXPRESSION_ERR: Raised if the expression is not legal
142  *   according to the rules of the <code>XPathEvaluator</code>

```

```

143 * <br>TYPE_ERR: Raised if the result cannot be converted to return the
144 * specified type.
145 * @exception DOMException
146 *   NAMESPACE_ERR: Raised if the expression contains namespace prefixes
147 *   which cannot be resolved by the specified
148 *   <code>XPathNSResolver</code>.
149 *   <br>WRONG_DOCUMENT_ERR: The Node is from a document that is not
150 *   supported by this <code>XPathEvaluator</code>.
151 *   <br>NOT_SUPPORTED_ERR: The Node is not a type permitted as an XPath
152 *   context node or the request type is not permitted by this
153 *   <code>XPathEvaluator</code>.
154 */
155 public Object evaluate(String expression,
156                       Node contextNode,
157                       XPathNSResolver resolver,
158                       short type,
159                       Object result)
160     throws XPathException, DOMException;
161
162 }

```

## 46.2 XPathException.java

Secuencia 473: org/w3c/dom/xpath/XPathException.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *

```

```

30  *
31  * Copyright (c) 2002 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.xpath;
43
44  /**
45   * A new exception has been created for exceptions specific to these XPath
46   * interfaces.
47   * <p>See also the <a href='http://www.w3.org/2002/08/WD-DOM-Level-3-XPath-20020820'>Document
48   * Object Model (DOM) Level 3 XPath Specification</a>.
49   */
49  public class XPathException extends RuntimeException {
50      public XPathException(short code, String message) {
51          super(message);
52          this.code = code;
53      }
54      public short    code;
55      // XPathExceptionCode
56      /**
57       * If the expression has a syntax error or otherwise is not a legal
58       * expression according to the rules of the specific
59       * <code>XPathEvaluator</code> or contains specialized extension
60       * functions or variables not supported by this implementation.
61       */
62      public static final short INVALID_EXPRESSION_ERR    = 1;
63      /**
64       * If the expression cannot be converted to return the specified type.
65       */
66      public static final short TYPE_ERR                  = 2;
67
68  }

```

### 46.3 XPathExpression.java

Secuencia 474: org/w3c/dom/xpath/XPathExpression.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *

```

```

10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2002 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.xpath;
43
44
45  import org.w3c.dom.Node;
46  import org.w3c.dom.DOMException;
47
48  /**
49   * The XPathExpression interface represents a parsed and resolved
50   * XPath expression.
51   * 

See also the Document
52   * Object Model (DOM) Level 3 XPath Specification.


53   */
54  public interface XPathExpression {
55      /**
56       * Evaluates this XPath expression and returns a result.
57       * @param contextNode The context is context node for the
58       * evaluation of this XPath expression. If the XPathEvaluator was
59       * obtained by casting the Document then this must be
60       * owned by the same document and must be a Document,
61       * Element, Attribute, Text,
62       * CDATASection, Comment,

```

```

62  * <code>ProcessingInstruction</code>, or <code>XPathNamespace</code>
63  * node.If the context node is a <code>Text</code> or a
64  * <code>CDATASection</code>, then the context is interpreted as the
65  * whole logical text node as seen by XPath, unless the node is empty
66  * in which case it may not serve as the XPath context.
67  * @param type If a specific <code>type</code> is specified, then the
68  * result will be coerced to return the specified type relying on
69  * XPath conversions and fail if the desired coercion is not possible.
70  * This must be one of the type codes of <code>XPathResult</code>.
71  * @param result The <code>result</code> specifies a specific result
72  * object which may be reused and returned by this method. If this is
73  * specified as <code>null</code>or the implementation does not reuse
74  * the specified result, a new result object will be constructed and
75  * returned.For XPath 1.0 results, this object will be of type
76  * <code>XPathResult</code>.
77  * @return The result of the evaluation of the XPath expression.For XPath
78  * 1.0 results, this object will be of type <code>XPathResult</code>.
79  * @exception XPathException
80  * TYPE_ERR: Raised if the result cannot be converted to return the
81  * specified type.
82  * @exception DOMException
83  * WRONG_DOCUMENT_ERR: The Node is from a document that is not supported
84  * by the XPathEvaluator that created this <code>XPathExpression</code>
85  * .
86  * <br>NOT_SUPPORTED_ERR: The Node is not a type permitted as an XPath
87  * context node or the request type is not permitted by this
88  * <code>XPathExpression</code>.
89  */
90  public Object evaluate(Node contextNode,
91                        short type,
92                        Object result)
93      throws XPathException, DOMException;
94
95  }

```

#### 46.4 XPathNSResolver.java

Secuencia 475: org/w3c/dom/xpath/XPathNSResolver.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *

```

```

16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  */
24
25  /*
26  *
27  *
28  *
29  *
30  *
31  * Copyright (c) 2002 World Wide Web Consortium,
32  * (Massachusetts Institute of Technology, Institut National de
33  * Recherche en Informatique et en Automatique, Keio University). All
34  * Rights Reserved. This program is distributed under the W3C's Software
35  * Intellectual Property License. This program is distributed in the
36  * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38  * PURPOSE.
39  * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40  */
41
42  package org.w3c.dom.xpath;
43
44
45  /**
46   * The XPathNSResolver interface permit prefix
47   * strings in the expression to be properly bound to
48   * namespaceURI strings. XPathEvaluator can
49   * construct an implementation of XPathNSResolver from a node,
50   * or the interface may be implemented by any application.
51   * 

See also the http://www.w3.org/2002/08/WD-DOM-Level-3-XPath-20020820 Document
52   * Object Model (DOM) Level 3 XPath Specification.


53   */
54  public interface XPathNSResolver {
55      /**
56       * Look up the namespace URI associated to the given namespace prefix. The
57       * XPath evaluator must never call this with a null or
58       * empty argument, because the result of doing this is undefined.
59       * @param prefix The prefix to look for.
60       * @return Returns the associated namespace URI or null if
61       *         none is found.
62       */
63      public String lookupNamespaceURI(String prefix);
64  }

```

## 46.5 XPathNamespace.java



Secuencia 476: org/w3c/dom/xpath/XPathNamespace.java

```
1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2002 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.xpath;
43
44
45 import org.w3c.dom.Element;
46 import org.w3c.dom.Node;
47
48 /**
49 * The XPathNamespace interface is returned by
50 * XPathResult interfaces to represent the XPath namespace node
51 * type that DOM lacks. There is no public constructor for this node type.
52 * Attempts to place it into a hierarchy or a NamedNodeMap result in a
```

```

53 * <code>DOMException</code> with the code <code>HIERARCHY_REQUEST_ERR</code>
54 * . This node is read only, so methods or setting of attributes that would
55 * mutate the node result in a DOMException with the code
56 * <code>NO_MODIFICATION_ALLOWED_ERR</code>.
57 * <p>The core specification describes attributes of the <code>Node</code>
58 * interface that are different for different node node types but does not
59 * describe <code>XPathNamespaceNode</code>, so here is a description of
60 * those attributes for this node type. All attributes of <code>Node</code>
61 * not described in this section have a <code>null</code> or
62 * <code>false</code> value.
63 * <p><code>ownerDocument</code> matches the <code>ownerDocument</code> of the
64 * <code>ownerElement</code> even if the element is later adopted.
65 * <p><code>prefix</code> is the prefix of the namespace represented by the
66 * node.
67 * <p><code>nodeName</code> is the same as <code>prefix</code>.
68 * <p><code>nodeType</code> is equal to <code>XPathNamespaceNode</code>.
69 * <p><code>namespaceURI</code> is the namespace URI of the namespace
70 * represented by the node.
71 * <p><code>adoptNode</code>, <code>cloneNode</code>, and
72 * <code>importNode</code> fail on this node type by raising a
73 * <code>DOMException</code> with the code <code>NOT_SUPPORTED_ERR</code>. In
74 * future versions of the XPath specification, the definition of a namespace
75 * node may be changed incompatibly, in which case incompatible changes to
76 * field values may be required to implement versions beyond XPath 1.0.
77 * <p>See also the <a href='http://www.w3.org/2002/08/WD-DOM-Level-3-XPath-20020820'>Document
    Object Model (DOM) Level 3 XPath Specification</a>.
78 */
79 public interface XPathNamespace extends Node {
80     // XPathNodeType
81     /**
82      * The node is a <code>Namespace</code>.
83      */
84     public static final short XPathNamespaceNode = 13;
85
86     /**
87      * The <code>Element</code> on which the namespace was in scope when it
88      * was requested. This does not change on a returned namespace node even
89      * if the document changes such that the namespace goes out of scope on
90      * that element and this node is no longer found there by XPath.
91      */
92     public Element getOwnerElement();
93
94 }

```

## 46.6 XPathResult.java

Secuencia 477: org/w3c/dom/xpath/XPathResult.java

```

1  /*
2  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
3  *
4  *
5  *
6  *

```

```
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 */
24
25 /*
26 *
27 *
28 *
29 *
30 *
31 * Copyright (c) 2002 World Wide Web Consortium,
32 * (Massachusetts Institute of Technology, Institut National de
33 * Recherche en Informatique et en Automatique, Keio University). All
34 * Rights Reserved. This program is distributed under the W3C's Software
35 * Intellectual Property License. This program is distributed in the
36 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even
37 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
38 * PURPOSE.
39 * See W3C License http://www.w3.org/Consortium/Legal/ for more details.
40 */
41
42 package org.w3c.dom.xpath;
43
44
45 import org.w3c.dom.Node;
46 import org.w3c.dom.DOMException;
47
48 /**
49 * The XPathResult interface represents the result of the
50 * evaluation of an XPath 1.0 expression within the context of a particular
51 * node. Since evaluation of an XPath expression can result in various
52 * result types, this object makes it possible to discover and manipulate
53 * the type and value of the result.
54 * <p>See also the <a href='http://www.w3.org/2002/08/WD-DOM-Level-3-XPath-20020820'>Document
55 *   Object Model (DOM) Level 3 XPath Specification</a>.
56 */
57 public interface XPathResult {
58     // XPathResultType
59     /**
```

```

59      * This code does not represent a specific type. An evaluation of an XPath
60      * expression will never produce this type. If this type is requested,
61      * then the evaluation returns whatever type naturally results from
62      * evaluation of the expression.
63      * <br>If the natural result is a node set when <code>ANY_TYPE</code> was
64      * requested, then <code>UNORDERED_NODE_ITERATOR_TYPE</code> is always
65      * the resulting type. Any other representation of a node set must be
66      * explicitly requested.
67      */
68      public static final short ANY_TYPE = 0;
69      /**
70       * The result is a number as defined by . Document modification does not
71       * invalidate the number, but may mean that reevaluation would not yield
72       * the same number.
73       */
74      public static final short NUMBER_TYPE = 1;
75      /**
76       * The result is a string as defined by . Document modification does not
77       * invalidate the string, but may mean that the string no longer
78       * corresponds to the current document.
79       */
80      public static final short STRING_TYPE = 2;
81      /**
82       * The result is a boolean as defined by . Document modification does not
83       * invalidate the boolean, but may mean that reevaluation would not
84       * yield the same boolean.
85       */
86      public static final short BOOLEAN_TYPE = 3;
87      /**
88       * The result is a node set as defined by that will be accessed
89       * iteratively, which may not produce nodes in a particular order.
90       * Document modification invalidates the iteration.
91       * <br>This is the default type returned if the result is a node set and
92       * <code>ANY_TYPE</code> is requested.
93       */
94      public static final short UNORDERED_NODE_ITERATOR_TYPE = 4;
95      /**
96       * The result is a node set as defined by that will be accessed
97       * iteratively, which will produce document-ordered nodes. Document
98       * modification invalidates the iteration.
99       */
100     public static final short ORDERED_NODE_ITERATOR_TYPE = 5;
101     /**
102      * The result is a node set as defined by that will be accessed as a
103      * snapshot list of nodes that may not be in a particular order.
104      * Document modification does not invalidate the snapshot but may mean
105      * that reevaluation would not yield the same snapshot and nodes in the
106      * snapshot may have been altered, moved, or removed from the document.
107      */
108     public static final short UNORDERED_NODE_SNAPSHOT_TYPE = 6;
109     /**
110      * The result is a node set as defined by that will be accessed as a
111      * snapshot list of nodes that will be in original document order.

```

```

112     * Document modification does not invalidate the snapshot but may mean
113     * that reevaluation would not yield the same snapshot and nodes in the
114     * snapshot may have been altered, moved, or removed from the document.
115     */
116     public static final short ORDERED_NODE_SNAPSHOT_TYPE = 7;
117     /**
118     * The result is a node set as defined by and will be accessed as a
119     * single node, which may be <code>null</code> if the node set is empty.
120     * Document modification does not invalidate the node, but may mean that
121     * the result node no longer corresponds to the current document. This
122     * is a convenience that permits optimization since the implementation
123     * can stop once any node in the in the resulting set has been found.
124     * <br>If there are more than one node in the actual result, the single
125     * node returned might not be the first in document order.
126     */
127     public static final short ANY_UNORDERED_NODE_TYPE = 8;
128     /**
129     * The result is a node set as defined by and will be accessed as a
130     * single node, which may be <code>null</code> if the node set is empty.
131     * Document modification does not invalidate the node, but may mean that
132     * the result node no longer corresponds to the current document. This
133     * is a convenience that permits optimization since the implementation
134     * can stop once the first node in document order of the resulting set
135     * has been found.
136     * <br>If there are more than one node in the actual result, the single
137     * node returned will be the first in document order.
138     */
139     public static final short FIRST_ORDERED_NODE_TYPE = 9;
140
141     /**
142     * A code representing the type of this result, as defined by the type
143     * constants.
144     */
145     public short getResultType();
146
147     /**
148     * The value of this number result. If the native double type of the DOM
149     * binding does not directly support the exact IEEE 754 result of the
150     * XPath expression, then it is up to the definition of the binding
151     * binding to specify how the XPath number is converted to the native
152     * binding number.
153     * @exception XPathException
154     *     TYPE_ERR: raised if <code>resultType</code> is not
155     *     <code>NUMBER_TYPE</code>.
156     */
157     public double getNumberValue()
158         throws XPathException;
159
160     /**
161     * The value of this string result.
162     * @exception XPathException
163     *     TYPE_ERR: raised if <code>resultType</code> is not
164     *     <code>STRING_TYPE</code>.

```

```

165     */
166     public String getStringValue()
167         throws XPathException;
168
169     /**
170     * The value of this boolean result.
171     * @exception XPathException
172     *     TYPE_ERR: raised if <code>resultType</code> is not
173     *     <code>BOOLEAN_TYPE</code>.
174     */
175     public boolean getBooleanValue()
176         throws XPathException;
177
178     /**
179     * The value of this single node result, which may be <code>null</code>.
180     * @exception XPathException
181     *     TYPE_ERR: raised if <code>resultType</code> is not
182     *     <code>ANY_UNORDERED_NODE_TYPE</code> or
183     *     <code>FIRST_ORDERED_NODE_TYPE</code>.
184     */
185     public Node getSingleNodeValue()
186         throws XPathException;
187
188     /**
189     * Signifies that the iterator has become invalid. True if
190     * <code>resultType</code> is <code>UNORDERED_NODE_ITERATOR_TYPE</code>
191     * or <code>ORDERED_NODE_ITERATOR_TYPE</code> and the document has been
192     * modified since this result was returned.
193     */
194     public boolean getInvalidIteratorState();
195
196     /**
197     * The number of nodes in the result snapshot. Valid values for
198     * snapshotItem indices are <code>0</code> to
199     * <code>snapshotLength-1</code> inclusive.
200     * @exception XPathException
201     *     TYPE_ERR: raised if <code>resultType</code> is not
202     *     <code>UNORDERED_NODE_SNAPSHOT_TYPE</code> or
203     *     <code>ORDERED_NODE_SNAPSHOT_TYPE</code>.
204     */
205     public int getSnapshotLength()
206         throws XPathException;
207
208     /**
209     * Iterates and returns the next node from the node set or
210     * <code>null</code> if there are no more nodes.
211     * @return Returns the next node.
212     * @exception XPathException
213     *     TYPE_ERR: raised if <code>resultType</code> is not
214     *     <code>UNORDERED_NODE_ITERATOR_TYPE</code> or
215     *     <code>ORDERED_NODE_ITERATOR_TYPE</code>.
216     * @exception DOMException
217     *     INVALID_STATE_ERR: The document has been mutated since the result was

```

```

218     *   returned.
219     */
220     public Node iterateNext()
221         throws XPathException, DOMException;
222
223     /**
224     * Returns the <code>index</code>th item in the snapshot collection. If
225     * <code>index</code> is greater than or equal to the number of nodes in
226     * the list, this method returns <code>null</code>. Unlike the iterator
227     * result, the snapshot does not become invalid, but may not correspond
228     * to the current document if it is mutated.
229     * @param index Index into the snapshot collection.
230     * @return The node at the <code>index</code>th position in the
231     *         <code>NodeList</code>, or <code>null</code> if that is not a valid
232     *         index.
233     * @exception XPathException
234     *         TYPE_ERR: raised if <code>resultType</code> is not
235     *         <code>UNORDERED_NODE_SNAPSHOT_TYPE</code> or
236     *         <code>ORDERED_NODE_SNAPSHOT_TYPE</code>.
237     */
238     public Node snapshotItem(int index)
239         throws XPathException;
240
241 }

```

## 47 Sax (org.xml.sax package)

### 47.1 AttributeList.java

Secuencia 478: org/xml/sax/AttributeList.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *

```

```
24  */
25
26  // SAX Attribute List Interface.
27  // http://www.saxproject.org
28  // No warranty; no copyright -- use this as you will.
29  // $Id: AttributeList.java,v 1.3 2004/11/03 22:44:51 jsuttor Exp $
30
31  package org.xml.sax;
32
33  /**
34   * Interface for an element's attribute specifications.
35   *
36   * <blockquote>
37   * <em>This module, both source code and documentation, is in the
38   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
39   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
40   * for further information.
41   * </blockquote>
42   *
43   * <p>This is the original SAX1 interface for reporting an element's
44   * attributes. Unlike the new {@link org.xml.sax.Attributes Attributes}
45   * interface, it does not support Namespace-related information.</p>
46   *
47   * <p>When an attribute list is supplied as part of a
48   * {@link org.xml.sax.DocumentHandler#startElement startElement}
49   * event, the list will return valid results only during the
50   * scope of the event; once the event handler returns control
51   * to the parser, the attribute list is invalid. To save a
52   * persistent copy of the attribute list, use the SAX1
53   * {@link org.xml.sax.helpers.AttributeListImpl AttributeListImpl}
54   * helper class.</p>
55   *
56   * <p>An attribute list includes only attributes that have been
57   * specified or defaulted: #IMPLIED attributes will not be included.</p>
58   *
59   * <p>There are two ways for the SAX application to obtain information
60   * from the AttributeList. First, it can iterate through the entire
61   * list:</p>
62   *
63   * <pre>
64   * public void startElement (String name, AttributeList atts) {
65   *     for (int i = 0; i < atts.getLength(); i++) {
66   *         String name = atts.getName(i);
67   *         String type = atts.getType(i);
68   *         String value = atts.getValue(i);
69   *         [...]
70   *     }
71   * }
72   * </pre>
73   *
74   * <p>(Note that the result of getLength() will be zero if there
75   * are no attributes.)
76   *
```



```

77  * <p>As an alternative, the application can request the value or
78  * type of specific attributes:</p>
79  *
80  * <pre>
81  * public void startElement (String name, AttributeList atts) {
82  *     String identifier = atts.getValue("id");
83  *     String label = atts.getValue("label");
84  *     [...]
85  * }
86  * </pre>
87  *
88  * @deprecated This interface has been replaced by the SAX2
89  *     {@link org.xml.sax.Attributes Attributes}
90  *     interface, which includes Namespace support.
91  * @since SAX 1.0
92  * @author David Megginson
93  * @see org.xml.sax.DocumentHandler#startElement startElement
94  * @see org.xml.sax.helpers.AttributeListImpl AttributeListImpl
95  */
96  public interface AttributeList {
97
98
99      ///////////////////////////////////////////////////
100     // Iteration methods.
101     ///////////////////////////////////////////////////
102
103
104     /**
105      * Return the number of attributes in this list.
106      *
107      * <p>The SAX parser may provide attributes in any
108      * arbitrary order, regardless of the order in which they were
109      * declared or specified. The number of attributes may be
110      * zero.</p>
111      *
112      * @return The number of attributes in the list.
113      */
114     public abstract int getLength ();
115
116
117     /**
118      * Return the name of an attribute in this list (by position).
119      *
120      * <p>The names must be unique: the SAX parser shall not include the
121      * same attribute twice. Attributes without values (those declared
122      * #IMPLIED without a value specified in the start tag) will be
123      * omitted from the list.</p>
124      *
125      * <p>If the attribute name has a namespace prefix, the prefix
126      * will still be attached.</p>
127      *
128      * @param i The index of the attribute in the list (starting at 0).
129      * @return The name of the indexed attribute, or null

```



```

183     * Return the type of an attribute in the list (by name).
184     *
185     * <p>The return value is the same as the return value for
186     * getType(int).</p>
187     *
188     * <p>If the attribute name has a namespace prefix in the document,
189     * the application must include the prefix here.</p>
190     *
191     * @param name The name of the attribute.
192     * @return The attribute type as a string, or null if no
193     *         such attribute exists.
194     * @see #getType(int)
195     */
196     public abstract String getType (String name);
197
198
199     /**
200     * Return the value of an attribute in the list (by name).
201     *
202     * <p>The return value is the same as the return value for
203     * getValue(int).</p>
204     *
205     * <p>If the attribute name has a namespace prefix in the document,
206     * the application must include the prefix here.</p>
207     *
208     * @param name the name of the attribute to return
209     * @return The attribute value as a string, or null if
210     *         no such attribute exists.
211     * @see #getValue(int)
212     */
213     public abstract String getValue (String name);
214
215 }
216
217 // end of AttributeList.java

```

## 47.2 Attributes.java

Secuencia 479: org/xml/sax/Attributes.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```

```
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  // Attributes.java - attribute list with Namespace support
27  // http://www.saxproject.org
28  // Written by David Megginson
29  // NO WARRANTY! This class is in the public domain.
30  // $Id: Attributes.java,v 1.2 2004/11/03 22:44:51 jsuttor Exp $
31
32  package org.xml.sax;
33
34
35  /**
36   * Interface for a list of XML attributes.
37   *
38   * <blockquote>
39   * <em>This module, both source code and documentation, is in the
40   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
41   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
42   * for further information.
43   * </blockquote>
44   *
45   * <p>This interface allows access to a list of attributes in
46   * three different ways:</p>
47   *
48   * <ol>
49   * <li>by attribute index;</li>
50   * <li>by Namespace-qualified name; or</li>
51   * <li>by qualified (prefixed) name.</li>
52   * </ol>
53   *
54   * <p>The list will not contain attributes that were declared
55   * #IMPLIED but not specified in the start tag. It will also not
56   * contain attributes used as Namespace declarations (xmlns*) unless
57   * the <code>http://xml.org/sax/features/namespace-prefixes</code>
58   * feature is set to <var>true</var> (it is <var>false</var> by
59   * default).
60   * Because SAX2 conforms to the original "Namespaces in XML"
61   * recommendation, it normally does not
62   * give namespace declaration attributes a namespace URI.
63   * </p>
64   *
65   * <p>Some SAX2 parsers may support using an optional feature flag
66   * (<code>http://xml.org/sax/features/xmlns-uris</code>) to request
67   * that those attributes be given URIs, conforming to a later
```

```

68  * backwards-incompatible revision of that recommendation. (The
69  * attribute's "local name" will be the prefix, or "xmlns" when
70  * defining a default element namespace.) For portability, handler
71  * code should always resolve that conflict, rather than requiring
72  * parsers that can change the setting of that feature flag. </p>
73  *
74  * <p>If the namespace-prefixes feature (see above) is
75  * <var>false</var>, access by qualified name may not be available; if
76  * the <code>http://xml.org/sax/features/namespaces</code> feature is
77  * <var>false</var>, access by Namespace-qualified names may not be
78  * available.</p>
79  *
80  * <p>This interface replaces the now-deprecated SAX1 {@link
81  * org.xml.sax.AttributeList AttributeList} interface, which does not
82  * contain Namespace support. In addition to Namespace support, it
83  * adds the <var>getIndex</var> methods (below).</p>
84  *
85  * <p>The order of attributes in the list is unspecified, and will
86  * vary from implementation to implementation.</p>
87  *
88  * @since SAX 2.0
89  * @author David Megginson
90  * @see org.xml.sax.helpers.AttributesImpl
91  * @see org.xml.sax.ext.DeclHandler#attributeDecl
92  */
93  public interface Attributes
94  {
95
96
97      ////////////////////////////////////////////
98      // Indexed access.
99      ////////////////////////////////////////////
100
101
102      /**
103       * Return the number of attributes in the list.
104       *
105       * <p>Once you know the number of attributes, you can iterate
106       * through the list.</p>
107       *
108       * @return The number of attributes in the list.
109       * @see #getURI(int)
110       * @see #getLocalName(int)
111       * @see #getQName(int)
112       * @see #getType(int)
113       * @see #getValue(int)
114       */
115      public abstract int getLength ();
116
117
118      /**
119       * Look up an attribute's Namespace URI by index.
120       *

```

```

121     * @param index The attribute index (zero-based).
122     * @return The Namespace URI, or the empty string if none
123     *         is available, or null if the index is out of
124     *         range.
125     * @see #getLength
126     */
127     public abstract String getURI (int index);
128
129
130     /**
131     * Look up an attribute's local name by index.
132     *
133     * @param index The attribute index (zero-based).
134     * @return The local name, or the empty string if Namespace
135     *         processing is not being performed, or null
136     *         if the index is out of range.
137     * @see #getLength
138     */
139     public abstract String getLocalName (int index);
140
141
142     /**
143     * Look up an attribute's XML qualified (prefixed) name by index.
144     *
145     * @param index The attribute index (zero-based).
146     * @return The XML qualified name, or the empty string
147     *         if none is available, or null if the index
148     *         is out of range.
149     * @see #getLength
150     */
151     public abstract String getQName (int index);
152
153
154     /**
155     * Look up an attribute's type by index.
156     *
157     * <p>The attribute type is one of the strings "CDATA", "ID",
158     * "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES",
159     * or "NOTATION" (always in upper case).</p>
160     *
161     * <p>If the parser has not read a declaration for the attribute,
162     * or if the parser does not report attribute types, then it must
163     * return the value "CDATA" as stated in the XML 1.0 Recommendation
164     * (clause 3.3.3, "Attribute-Value Normalization").</p>
165     *
166     * <p>For an enumerated attribute that is not a notation, the
167     * parser will report the type as "NMTOKEN".</p>
168     *
169     * @param index The attribute index (zero-based).
170     * @return The attribute's type as a string, or null if the
171     *         index is out of range.
172     * @see #getLength
173     */

```

```

174     public abstract String getType (int index);
175
176
177     /**
178      * Look up an attribute's value by index.
179      *
180      * <p>If the attribute value is a list of tokens (IDREFS,
181      * ENTITIES, or NMTOKENS), the tokens will be concatenated
182      * into a single string with each token separated by a
183      * single space.</p>
184      *
185      * @param index The attribute index (zero-based).
186      * @return The attribute's value as a string, or null if the
187      *         index is out of range.
188      * @see #getLength
189      */
190     public abstract String getValue (int index);
191
192
193
194     //////////////////////////////////////
195     // Name-based query.
196     //////////////////////////////////////
197
198
199     /**
200      * Look up the index of an attribute by Namespace name.
201      *
202      * @param uri The Namespace URI, or the empty string if
203      *         the name has no Namespace URI.
204      * @param localName The attribute's local name.
205      * @return The index of the attribute, or -1 if it does not
206      *         appear in the list.
207      */
208     public int getIndex (String uri, String localName);
209
210
211     /**
212      * Look up the index of an attribute by XML qualified (prefixed) name.
213      *
214      * @param qName The qualified (prefixed) name.
215      * @return The index of the attribute, or -1 if it does not
216      *         appear in the list.
217      */
218     public int getIndex (String qName);
219
220
221     /**
222      * Look up an attribute's type by Namespace name.
223      *
224      * <p>See {@link #getType(int) getType(int)} for a description
225      * of the possible types.</p>
226      *

```

```
227     * @param uri The Namespace URI, or the empty String if the
228     *           name has no Namespace URI.
229     * @param localName The local name of the attribute.
230     * @return The attribute type as a string, or null if the
231     *         attribute is not in the list or if Namespace
232     *         processing is not being performed.
233     */
234     public abstract String getType (String uri, String localName);
235
236
237     /**
238     * Look up an attribute's type by XML qualified (prefixed) name.
239     *
240     * <p>See {@link #getType(int) getType(int)} for a description
241     * of the possible types.</p>
242     *
243     * @param qName The XML qualified name.
244     * @return The attribute type as a string, or null if the
245     *         attribute is not in the list or if qualified names
246     *         are not available.
247     */
248     public abstract String getType (String qName);
249
250
251     /**
252     * Look up an attribute's value by Namespace name.
253     *
254     * <p>See {@link #getValue(int) getValue(int)} for a description
255     * of the possible values.</p>
256     *
257     * @param uri The Namespace URI, or the empty String if the
258     *           name has no Namespace URI.
259     * @param localName The local name of the attribute.
260     * @return The attribute value as a string, or null if the
261     *         attribute is not in the list.
262     */
263     public abstract String getValue (String uri, String localName);
264
265
266     /**
267     * Look up an attribute's value by XML qualified (prefixed) name.
268     *
269     * <p>See {@link #getValue(int) getValue(int)} for a description
270     * of the possible values.</p>
271     *
272     * @param qName The XML qualified name.
273     * @return The attribute value as a string, or null if the
274     *         attribute is not in the list or if qualified names
275     *         are not available.
276     */
277     public abstract String getValue (String qName);
278
279 }
```



```
280
281 // end of Attributes.java
```

### 47.3 ContentHandler.java

Secuencia 480: org/xml/sax/ContentHandler.java

```
1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // ContentHandler.java - handle main document content.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the public domain.
30 // $Id: ContentHandler.java,v 1.2 2004/11/03 22:44:51 jsuttor Exp $
31
32 package org.xml.sax;
33
34
35 /**
36 * Receive notification of the logical content of a document.
37 *
38 * <blockquote>
39 * <em>This module, both source code and documentation, is in the
40 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
41 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
42 * for further information.
43 * </blockquote>
44 *
45 * <p>This is the main interface that most SAX applications
46 * implement: if the application needs to be informed of basic parsing
47 * events, it implements this interface and registers an instance with
```

```

48 * the SAX parser using the {@link org.xml.sax.XMLReader#setContentHandler
49 * setContentHandler} method. The parser uses the instance to report
50 * basic document-related events like the start and end of elements
51 * and character data.</p>
52 *
53 * <p>The order of events in this interface is very important, and
54 * mirrors the order of information in the document itself. For
55 * example, all of an element's content (character data, processing
56 * instructions, and/or subelements) will appear, in order, between
57 * the startElement event and the corresponding endElement event.</p>
58 *
59 * <p>This interface is similar to the now-deprecated SAX 1.0
60 * DocumentHandler interface, but it adds support for Namespaces
61 * and for reporting skipped entities (in non-validating XML
62 * processors).</p>
63 *
64 * <p>Implementors should note that there is also a
65 * <code>ContentHandler</code> class in the <code>java.net</code>
66 * package; that means that it's probably a bad idea to do</p>
67 *
68 * <pre>import java.net.*;
69 * import org.xml.sax.*;
70 * </pre>
71 *
72 * <p>In fact, "import ...*" is usually a sign of sloppy programming
73 * anyway, so the user should consider this a feature rather than a
74 * bug.</p>
75 *
76 * @since SAX 2.0
77 * @author David Megginson
78 * @see org.xml.sax.XMLReader
79 * @see org.xml.sax.DTDHandler
80 * @see org.xml.sax.ErrorHandler
81 */
82 public interface ContentHandler
83 {
84
85     /**
86      * Receive an object for locating the origin of SAX document events.
87      *
88      * <p>SAX parsers are strongly encouraged (though not absolutely
89      * required) to supply a locator: if it does so, it must supply
90      * the locator to the application by invoking this method before
91      * invoking any of the other methods in the ContentHandler
92      * interface.</p>
93      *
94      * <p>The locator allows the application to determine the end
95      * position of any document-related event, even if the parser is
96      * not reporting an error. Typically, the application will
97      * use this information for reporting its own errors (such as
98      * character content that does not match an application's
99      * business rules). The information returned by the locator
100      * is probably not sufficient for use with a search engine.</p>

```

```

101      *
102      * <p>Note that the locator will return correct information only
103      * during the invocation SAX event callbacks after
104      * {@link #startDocument startDocument} returns and before
105      * {@link #endDocument endDocument} is called. The
106      * application should not attempt to use it at any other time.</p>
107      *
108      * @param locator an object that can return the location of
109      *               any SAX document event
110      * @see org.xml.sax.Locator
111      */
112     public void setDocumentLocator (Locator locator);
113
114
115     /**
116      * Receive notification of the beginning of a document.
117      *
118      * <p>The SAX parser will invoke this method only once, before any
119      * other event callbacks (except for {@link #setDocumentLocator
120      * setDocumentLocator}).</p>
121      *
122      * @throws org.xml.sax.SAXException any SAX exception, possibly
123      *               wrapping another exception
124      * @see #endDocument
125      */
126     public void startDocument ()
127         throws SAXException;
128
129
130     /**
131      * Receive notification of the end of a document.
132      *
133      * <p><strong>There is an apparent contradiction between the
134      * documentation for this method and the documentation for {@link
135      * org.xml.sax.ErrorHandler#fatalError}. Until this ambiguity is
136      * resolved in a future major release, clients should make no
137      * assumptions about whether endDocument() will or will not be
138      * invoked when the parser has reported a fatalError() or thrown
139      * an exception.</strong></p>
140      *
141      * <p>The SAX parser will invoke this method only once, and it will
142      * be the last method invoked during the parse. The parser shall
143      * not invoke this method until it has either abandoned parsing
144      * (because of an unrecoverable error) or reached the end of
145      * input.</p>
146      *
147      * @throws org.xml.sax.SAXException any SAX exception, possibly
148      *               wrapping another exception
149      * @see #startDocument
150      */
151     public void endDocument()
152         throws SAXException;
153

```

```

154
155 /**
156  * Begin the scope of a prefix-URI Namespace mapping.
157  *
158  * <p>The information from this event is not necessary for
159  * normal Namespace processing: the SAX XML reader will
160  * automatically replace prefixes for element and attribute
161  * names when the <code>http://xml.org/sax/features/namespaces</code>
162  * feature is <var>true</var> (the default).</p>
163  *
164  * <p>There are cases, however, when applications need to
165  * use prefixes in character data or in attribute values,
166  * where they cannot safely be expanded automatically; the
167  * start/endPrefixMapping event supplies the information
168  * to the application to expand prefixes in those contexts
169  * itself, if necessary.</p>
170  *
171  * <p>Note that start/endPrefixMapping events are not
172  * guaranteed to be properly nested relative to each other:
173  * all startPrefixMapping events will occur immediately before the
174  * corresponding {@link #startElement startElement} event,
175  * and all {@link #endPrefixMapping endPrefixMapping}
176  * events will occur immediately after the corresponding
177  * {@link #endElement endElement} event,
178  * but their order is not otherwise
179  * guaranteed.</p>
180  *
181  * <p>There should never be start/endPrefixMapping events for the
182  * "xml" prefix, since it is predeclared and immutable.</p>
183  *
184  * @param prefix the Namespace prefix being declared.
185  *   An empty string is used for the default element namespace,
186  *   which has no prefix.
187  * @param uri the Namespace URI the prefix is mapped to
188  * @throws org.xml.sax.SAXException the client may throw
189  *   an exception during processing
190  * @see #endPrefixMapping
191  * @see #startElement
192  */
193 public void startPrefixMapping (String prefix, String uri)
194     throws SAXException;
195
196
197 /**
198  * End the scope of a prefix-URI mapping.
199  *
200  * <p>See {@link #startPrefixMapping startPrefixMapping} for
201  * details. These events will always occur immediately after the
202  * corresponding {@link #endElement endElement} event, but the order of
203  * {@link #endPrefixMapping endPrefixMapping} events is not otherwise
204  * guaranteed.</p>
205  *
206  * @param prefix the prefix that was being mapped.

```

```
207 * This is the empty string when a default mapping scope ends.
208 * @throws org.xml.sax.SAXException the client may throw
209 *         an exception during processing
210 * @see #startPrefixMapping
211 * @see #endElement
212 */
213 public void endPrefixMapping (String prefix)
214     throws SAXException;
215
216
217 /**
218 * Receive notification of the beginning of an element.
219 *
220 * <p>The Parser will invoke this method at the beginning of every
221 * element in the XML document; there will be a corresponding
222 * {@link #endElement endElement} event for every startElement event
223 * (even when the element is empty). All of the element's content will be
224 * reported, in order, before the corresponding endElement
225 * event.</p>
226 *
227 * <p>This event allows up to three name components for each
228 * element:</p>
229 *
230 * <ol>
231 * <li>the Namespace URI;</li>
232 * <li>the local name; and</li>
233 * <li>the qualified (prefixed) name.</li>
234 * </ol>
235 *
236 * <p>Any or all of these may be provided, depending on the
237 * values of the <var>http://xml.org/sax/features/namespaces</var>
238 * and the <var>http://xml.org/sax/features/namespace-prefixes</var>
239 * properties:</p>
240 *
241 * <ul>
242 * <li>the Namespace URI and local name are required when
243 * the namespaces property is <var>true</var> (the default), and are
244 * optional when the namespaces property is <var>false</var> (if one is
245 * specified, both must be);</li>
246 * <li>the qualified name is required when the namespace-prefixes property
247 * is <var>true</var>, and is optional when the namespace-prefixes property
248 * is <var>false</var> (the default).</li>
249 * </ul>
250 *
251 * <p>Note that the attribute list provided will contain only
252 * attributes with explicit values (specified or defaulted):
253 * #IMPLIED attributes will be omitted. The attribute list
254 * will contain attributes used for Namespace declarations
255 * (xmlns* attributes) only if the
256 * <code>http://xml.org/sax/features/namespace-prefixes</code>
257 * property is true (it is false by default, and support for a
258 * true value is optional).</p>
259 *
```

```

260 * <p>Like {@link #characters characters()}, attribute values may have
261 * characters that need more than one <code>char</code> value. </p>
262 *
263 * @param uri the Namespace URI, or the empty string if the
264 *     element has no Namespace URI or if Namespace
265 *     processing is not being performed
266 * @param localName the local name (without prefix), or the
267 *     empty string if Namespace processing is not being
268 *     performed
269 * @param qName the qualified name (with prefix), or the
270 *     empty string if qualified names are not available
271 * @param atts the attributes attached to the element. If
272 *     there are no attributes, it shall be an empty
273 *     Attributes object. The value of this object after
274 *     startElement returns is undefined
275 * @throws org.xml.sax.SAXException any SAX exception, possibly
276 *     wrapping another exception
277 * @see #endElement
278 * @see org.xml.sax.Attributes
279 * @see org.xml.sax.helpers.AttributesImpl
280 */
281 public void startElement (String uri, String localName,
282     String qName, Attributes atts)
283     throws SAXException;
284
285
286 /**
287 * Receive notification of the end of an element.
288 *
289 * <p>The SAX parser will invoke this method at the end of every
290 * element in the XML document; there will be a corresponding
291 * {@link #startElement startElement} event for every endElement
292 * event (even when the element is empty).</p>
293 *
294 * <p>For information on the names, see startElement.</p>
295 *
296 * @param uri the Namespace URI, or the empty string if the
297 *     element has no Namespace URI or if Namespace
298 *     processing is not being performed
299 * @param localName the local name (without prefix), or the
300 *     empty string if Namespace processing is not being
301 *     performed
302 * @param qName the qualified XML name (with prefix), or the
303 *     empty string if qualified names are not available
304 * @throws org.xml.sax.SAXException any SAX exception, possibly
305 *     wrapping another exception
306 */
307 public void endElement (String uri, String localName,
308     String qName)
309     throws SAXException;
310
311
312 /**

```

```

313     * Receive notification of character data.
314     *
315     * <p>The Parser will call this method to report each chunk of
316     * character data. SAX parsers may return all contiguous character
317     * data in a single chunk, or they may split it into several
318     * chunks; however, all of the characters in any single event
319     * must come from the same external entity so that the Locator
320     * provides useful information.</p>
321     *
322     * <p>The application must not attempt to read from the array
323     * outside of the specified range.</p>
324     *
325     * <p>Individual characters may consist of more than one Java
326     * <code>char</code> value. There are two important cases where this
327     * happens, because characters can't be represented in just sixteen bits.
328     * In one case, characters are represented in a <em>Surrogate Pair</em>,
329     * using two special Unicode values. Such characters are in the so-called
330     * "Astral Planes", with a code point above U+FFFF. A second case involves
331     * composite characters, such as a base character combining with one or
332     * more accent characters. </p>
333     *
334     * <p>Your code should not assume that algorithms using
335     * <code>char</code>-at-a-time idioms will be working in character
336     * units; in some cases they will split characters. This is relevant
337     * wherever XML permits arbitrary characters, such as attribute values,
338     * processing instruction data, and comments as well as in data reported
339     * from this method. It's also generally relevant whenever Java code
340     * manipulates internationalized text; the issue isn't unique to XML.</p>
341     *
342     * <p>Note that some parsers will report whitespace in element
343     * content using the {@link #ignoreableWhitespace ignoreableWhitespace}
344     * method rather than this one (validating parsers <em>must</em>
345     * do so).</p>
346     *
347     * @param ch the characters from the XML document
348     * @param start the start position in the array
349     * @param length the number of characters to read from the array
350     * @throws org.xml.sax.SAXException any SAX exception, possibly
351     *         wrapping another exception
352     * @see #ignoreableWhitespace
353     * @see org.xml.sax.Locator
354     */
355     public void characters (char ch[], int start, int length)
356         throws SAXException;
357
358
359     /**
360     * Receive notification of ignorable whitespace in element content.
361     *
362     * <p>Validating Parsers must use this method to report each chunk
363     * of whitespace in element content (see the W3C XML 1.0
364     * recommendation, section 2.10): non-validating parsers may also
365     * use this method if they are capable of parsing and using

```

```

366     * content models.</p>
367     *
368     * <p>SAX parsers may return all contiguous whitespace in a single
369     * chunk, or they may split it into several chunks; however, all of
370     * the characters in any single event must come from the same
371     * external entity, so that the Locator provides useful
372     * information.</p>
373     *
374     * <p>The application must not attempt to read from the array
375     * outside of the specified range.</p>
376     *
377     * @param ch the characters from the XML document
378     * @param start the start position in the array
379     * @param length the number of characters to read from the array
380     * @throws org.xml.sax.SAXException any SAX exception, possibly
381     *         wrapping another exception
382     * @see #characters
383     */
384     public void ignorableWhitespace (char ch[], int start, int length)
385         throws SAXException;
386
387
388     /**
389     * Receive notification of a processing instruction.
390     *
391     * <p>The Parser will invoke this method once for each processing
392     * instruction found: note that processing instructions may occur
393     * before or after the main document element.</p>
394     *
395     * <p>A SAX parser must never report an XML declaration (XML 1.0,
396     * section 2.8) or a text declaration (XML 1.0, section 4.3.1)
397     * using this method.</p>
398     *
399     * <p>Like {@link #characters characters()}, processing instruction
400     * data may have characters that need more than one <code>char</code>
401     * value. </p>
402     *
403     * @param target the processing instruction target
404     * @param data the processing instruction data, or null if
405     *         none was supplied. The data does not include any
406     *         whitespace separating it from the target
407     * @throws org.xml.sax.SAXException any SAX exception, possibly
408     *         wrapping another exception
409     */
410     public void processingInstruction (String target, String data)
411         throws SAXException;
412
413
414     /**
415     * Receive notification of a skipped entity.
416     * This is not called for entity references within markup constructs
417     * such as element start tags or markup declarations. (The XML
418     * recommendation requires reporting skipped external entities.

```



```

419     * SAX also reports internal entity expansion/non-expansion, except
420     * within markup constructs.)
421     *
422     * <p>The Parser will invoke this method each time the entity is
423     * skipped. Non-validating processors may skip entities if they
424     * have not seen the declarations (because, for example, the
425     * entity was declared in an external DTD subset). All processors
426     * may skip external entities, depending on the values of the
427     * <code>http://xml.org/sax/features/external-general-entities</code>
428     * and the
429     * <code>http://xml.org/sax/features/external-parameter-entities</code>
430     * properties.</p>
431     *
432     * @param name the name of the skipped entity. If it is a
433     *       parameter entity, the name will begin with '%', and if
434     *       it is the external DTD subset, it will be the string
435     *       "[dtd]"
436     * @throws org.xml.sax.SAXException any SAX exception, possibly
437     *       wrapping another exception
438     */
439     public void skippedEntity (String name)
440         throws SAXException;
441 }
442
443 // end of ContentHandler.java

```

#### 47.4 DTDHandler.java

Secuencia 481: org/xml/sax/DTDHandler.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */

```

```

25
26 // SAX DTD handler.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: DTDHandler.java,v 1.2 2004/11/03 22:44:51 jsuttor Exp $
30
31 package org.xml.sax;
32
33 /**
34  * Receive notification of basic DTD-related events.
35  *
36  * <blockquote>
37  * <em>This module, both source code and documentation, is in the
38  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
39  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
40  * for further information.
41  * </blockquote>
42  *
43  * <p>If a SAX application needs information about notations and
44  * unparsed entities, then the application implements this
45  * interface and registers an instance with the SAX parser using
46  * the parser's setDTDHandler method. The parser uses the
47  * instance to report notation and unparsed entity declarations to
48  * the application.</p>
49  *
50  * <p>Note that this interface includes only those DTD events that
51  * the XML recommendation <em>requires</em> processors to report:
52  * notation and unparsed entity declarations.</p>
53  *
54  * <p>The SAX parser may report these events in any order, regardless
55  * of the order in which the notations and unparsed entities were
56  * declared; however, all DTD events must be reported after the
57  * document handler's startDocument event, and before the first
58  * startElement event.
59  * (If the {@link org.xml.sax.ext.LexicalHandler LexicalHandler} is
60  * used, these events must also be reported before the endDTD event.)
61  * </p>
62  *
63  * <p>It is up to the application to store the information for
64  * future use (perhaps in a hash table or object tree).
65  * If the application encounters attributes of type "NOTATION",
66  * "ENTITY", or "ENTITIES", it can use the information that it
67  * obtained through this interface to find the entity and/or
68  * notation corresponding with the attribute value.</p>
69  *
70  * @since SAX 1.0
71  * @author David Megginson
72  * @see org.xml.sax.XMLReader#setDTDHandler
73  */
74 public interface DTDHandler {
75
76
77     /**

```

```

78      * Receive notification of a notation declaration event.
79      *
80      * <p>It is up to the application to record the notation for later
81      * reference, if necessary;
82      * notations may appear as attribute values and in unparsed entity
83      * declarations, and are sometime used with processing instruction
84      * target names.</p>
85      *
86      * <p>At least one of publicId and systemId must be non-null.
87      * If a system identifier is present, and it is a URL, the SAX
88      * parser must resolve it fully before passing it to the
89      * application through this event.</p>
90      *
91      * <p>There is no guarantee that the notation declaration will be
92      * reported before any unparsed entities that use it.</p>
93      *
94      * @param name The notation name.
95      * @param publicId The notation's public identifier, or null if
96      *     none was given.
97      * @param systemId The notation's system identifier, or null if
98      *     none was given.
99      * @exception org.xml.sax.SAXException Any SAX exception, possibly
100     *     wrapping another exception.
101     * @see #unparsedEntityDecl
102     * @see org.xml.sax.Attributes
103     */
104     public abstract void notationDecl (String name,
105                                       String publicId,
106                                       String systemId)
107         throws SAXException;
108
109
110     /**
111     * Receive notification of an unparsed entity declaration event.
112     *
113     * <p>Note that the notation name corresponds to a notation
114     * reported by the {@link #notationDecl notationDecl} event.
115     * It is up to the application to record the entity for later
116     * reference, if necessary;
117     * unparsed entities may appear as attribute values.
118     * </p>
119     *
120     * <p>If the system identifier is a URL, the parser must resolve it
121     * fully before passing it to the application.</p>
122     *
123     * @exception org.xml.sax.SAXException Any SAX exception, possibly
124     *     wrapping another exception.
125     * @param name The unparsed entity's name.
126     * @param publicId The entity's public identifier, or null if none
127     *     was given.
128     * @param systemId The entity's system identifier.
129     * @param notationName The name of the associated notation.
130     * @see #notationDecl

```

```

131     * @see org.xml.sax.Attributes
132     */
133     public abstract void unparsedEntityDecl (String name,
134                                             String publicId,
135                                             String systemId,
136                                             String notationName)
137         throws SAXException;
138
139 }
140
141 // end of DTDHandler.java

```

## 47.5 DocumentHandler.java

Secuencia 482: org/xml/sax/DocumentHandler.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX document handler.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: DocumentHandler.java,v 1.2 2004/11/03 22:44:51 jsuttor Exp $
30
31 package org.xml.sax;
32
33 /**
34  * Receive notification of general document events.
35  *
36  * <blockquote>
37  * <em>This module, both source code and documentation, is in the
38  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>

```

```

39 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
40 * for further information.
41 * </blockquote>
42 *
43 * <p>This was the main event-handling interface for SAX1; in
44 * SAX2, it has been replaced by {@link org.xml.sax.ContentHandler
45 * ContentHandler}, which provides Namespace support and reporting
46 * of skipped entities. This interface is included in SAX2 only
47 * to support legacy SAX1 applications.</p>
48 *
49 * <p>The order of events in this interface is very important, and
50 * mirrors the order of information in the document itself. For
51 * example, all of an element's content (character data, processing
52 * instructions, and/or subelements) will appear, in order, between
53 * the startElement event and the corresponding endElement event.</p>
54 *
55 * <p>Application writers who do not want to implement the entire
56 * interface can derive a class from HandlerBase, which implements
57 * the default functionality; parser writers can instantiate
58 * HandlerBase to obtain a default handler. The application can find
59 * the location of any document event using the Locator interface
60 * supplied by the Parser through the setDocumentLocator method.</p>
61 *
62 * @deprecated This interface has been replaced by the SAX2
63 *             {@link org.xml.sax.ContentHandler ContentHandler}
64 *             interface, which includes Namespace support.
65 * @since SAX 1.0
66 * @author David Megginson
67 * @see org.xml.sax.Parser#setDocumentHandler
68 * @see org.xml.sax.Locator
69 * @see org.xml.sax.HandlerBase
70 */
71 public interface DocumentHandler {
72
73
74     /**
75      * Receive an object for locating the origin of SAX document events.
76      *
77      * <p>SAX parsers are strongly encouraged (though not absolutely
78      * required) to supply a locator: if it does so, it must supply
79      * the locator to the application by invoking this method before
80      * invoking any of the other methods in the DocumentHandler
81      * interface.</p>
82      *
83      * <p>The locator allows the application to determine the end
84      * position of any document-related event, even if the parser is
85      * not reporting an error. Typically, the application will
86      * use this information for reporting its own errors (such as
87      * character content that does not match an application's
88      * business rules). The information returned by the locator
89      * is probably not sufficient for use with a search engine.</p>
90      *
91      * <p>Note that the locator will return correct information only

```

```
92      * during the invocation of the events in this interface. The
93      * application should not attempt to use it at any other time.</p>
94      *
95      * @param locator An object that can return the location of
96      *               any SAX document event.
97      * @see org.xml.sax.Locator
98      */
99      public abstract void setDocumentLocator (Locator locator);
100
101
102      /**
103      * Receive notification of the beginning of a document.
104      *
105      * <p>The SAX parser will invoke this method only once, before any
106      * other methods in this interface or in DTDHandler (except for
107      * setDocumentLocator).</p>
108      *
109      * @exception org.xml.sax.SAXException Any SAX exception, possibly
110      *               wrapping another exception.
111      */
112      public abstract void startDocument ()
113          throws SAXException;
114
115
116      /**
117      * Receive notification of the end of a document.
118      *
119      * <p>The SAX parser will invoke this method only once, and it will
120      * be the last method invoked during the parse. The parser shall
121      * not invoke this method until it has either abandoned parsing
122      * (because of an unrecoverable error) or reached the end of
123      * input.</p>
124      *
125      * @exception org.xml.sax.SAXException Any SAX exception, possibly
126      *               wrapping another exception.
127      */
128      public abstract void endDocument ()
129          throws SAXException;
130
131
132      /**
133      * Receive notification of the beginning of an element.
134      *
135      * <p>The Parser will invoke this method at the beginning of every
136      * element in the XML document; there will be a corresponding
137      * endElement() event for every startElement() event (even when the
138      * element is empty). All of the element's content will be
139      * reported, in order, before the corresponding endElement()
140      * event.</p>
141      *
142      * <p>If the element name has a namespace prefix, the prefix will
143      * still be attached. Note that the attribute list provided will
144      * contain only attributes with explicit values (specified or
```

```
145     * defaulted): #IMPLIED attributes will be omitted.</p>
146     *
147     * @param name The element type name.
148     * @param atts The attributes attached to the element, if any.
149     * @exception org.xml.sax.SAXException Any SAX exception, possibly
150     *         wrapping another exception.
151     * @see #endElement
152     * @see org.xml.sax.AttributeList
153     */
154     public abstract void startElement (String name, AttributeList atts)
155         throws SAXException;
156
157
158     /**
159     * Receive notification of the end of an element.
160     *
161     * <p>The SAX parser will invoke this method at the end of every
162     * element in the XML document; there will be a corresponding
163     * startElement() event for every endElement() event (even when the
164     * element is empty).</p>
165     *
166     * <p>If the element name has a namespace prefix, the prefix will
167     * still be attached to the name.</p>
168     *
169     * @param name The element type name
170     * @exception org.xml.sax.SAXException Any SAX exception, possibly
171     *         wrapping another exception.
172     */
173     public abstract void endElement (String name)
174         throws SAXException;
175
176
177     /**
178     * Receive notification of character data.
179     *
180     * <p>The Parser will call this method to report each chunk of
181     * character data. SAX parsers may return all contiguous character
182     * data in a single chunk, or they may split it into several
183     * chunks; however, all of the characters in any single event
184     * must come from the same external entity, so that the Locator
185     * provides useful information.</p>
186     *
187     * <p>The application must not attempt to read from the array
188     * outside of the specified range.</p>
189     *
190     * <p>Note that some parsers will report whitespace using the
191     * ignorableWhitespace() method rather than this one (validating
192     * parsers must do so).</p>
193     *
194     * @param ch The characters from the XML document.
195     * @param start The start position in the array.
196     * @param length The number of characters to read from the array.
197     * @exception org.xml.sax.SAXException Any SAX exception, possibly
```

```
198     *           wrapping another exception.
199     * @see #ignorableWhitespace
200     * @see org.xml.sax Locator
201     */
202     public abstract void characters (char ch[], int start, int length)
203         throws SAXException;
204
205
206     /**
207     * Receive notification of ignorable whitespace in element content.
208     *
209     * <p>Validating Parsers must use this method to report each chunk
210     * of ignorable whitespace (see the W3C XML 1.0 recommendation,
211     * section 2.10): non-validating parsers may also use this method
212     * if they are capable of parsing and using content models.</p>
213     *
214     * <p>SAX parsers may return all contiguous whitespace in a single
215     * chunk, or they may split it into several chunks; however, all of
216     * the characters in any single event must come from the same
217     * external entity, so that the Locator provides useful
218     * information.</p>
219     *
220     * <p>The application must not attempt to read from the array
221     * outside of the specified range.</p>
222     *
223     * @param ch The characters from the XML document.
224     * @param start The start position in the array.
225     * @param length The number of characters to read from the array.
226     * @exception org.xml.sax.SAXException Any SAX exception, possibly
227     *           wrapping another exception.
228     * @see #characters
229     */
230     public abstract void ignorableWhitespace (char ch[], int start, int length)
231         throws SAXException;
232
233
234     /**
235     * Receive notification of a processing instruction.
236     *
237     * <p>The Parser will invoke this method once for each processing
238     * instruction found: note that processing instructions may occur
239     * before or after the main document element.</p>
240     *
241     * <p>A SAX parser should never report an XML declaration (XML 1.0,
242     * section 2.8) or a text declaration (XML 1.0, section 4.3.1)
243     * using this method.</p>
244     *
245     * @param target The processing instruction target.
246     * @param data The processing instruction data, or null if
247     *           none was supplied.
248     * @exception org.xml.sax.SAXException Any SAX exception, possibly
249     *           wrapping another exception.
250     */
```



```
251     public abstract void processingInstruction (String target, String data)
252         throws SAXException;
253
254 }
255
256 // end of DocumentHandler.java
```

## 47.6 EntityResolver.java

Secuencia 483: org/xml/sax/EntityResolver.java

```
1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX entity resolver.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: EntityResolver.java,v 1.2 2004/11/03 22:44:52 jsuttor Exp $
30
31 package org.xml.sax;
32
33 import java.io.IOException;
34
35
36 /**
37 * Basic interface for resolving entities.
38 *
39 * <blockquote>
40 * <em>This module, both source code and documentation, is in the
41 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
42 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
43 * for further information.
```

```

44 * </blockquote>
45 *
46 * <p>If a SAX application needs to implement customized handling
47 * for external entities, it must implement this interface and
48 * register an instance with the SAX driver using the
49 * {@link org.xml.sax.XMLReader#setEntityResolver setEntityResolver}
50 * method.</p>
51 *
52 * <p>The XML reader will then allow the application to intercept any
53 * external entities (including the external DTD subset and external
54 * parameter entities, if any) before including them.</p>
55 *
56 * <p>Many SAX applications will not need to implement this interface,
57 * but it will be especially useful for applications that build
58 * XML documents from databases or other specialised input sources,
59 * or for applications that use URI types other than URLs.</p>
60 *
61 * <p>The following resolver would provide the application
62 * with a special character stream for the entity with the system
63 * identifier "http://www.myhost.com/today":</p>
64 *
65 * <pre>
66 * import org.xml.sax.EntityResolver;
67 * import org.xml.sax.InputSource;
68 *
69 * public class MyResolver implements EntityResolver {
70 *     public InputSource resolveEntity (String publicId, String systemId)
71 *     {
72 *         if (systemId.equals("http://www.myhost.com/today")) {
73 *             // return a special input source
74 *             MyReader reader = new MyReader();
75 *             return new InputSource(reader);
76 *         } else {
77 *             // use the default behaviour
78 *             return null;
79 *         }
80 *     }
81 * }
82 * </pre>
83 *
84 * <p>The application can also use this interface to redirect system
85 * identifiers to local URIs or to look up replacements in a catalog
86 * (possibly by using the public identifier).</p>
87 *
88 * @since SAX 1.0
89 * @author David Megginson
90 * @see org.xml.sax.XMLReader#setEntityResolver
91 * @see org.xml.sax.InputSource
92 */
93 public interface EntityResolver {
94
95
96     /**

```

```

97      * Allow the application to resolve external entities.
98      *
99      * <p>The parser will call this method before opening any external
100     * entity except the top-level document entity. Such entities include
101     * the external DTD subset and external parameter entities referenced
102     * within the DTD (in either case, only if the parser reads external
103     * parameter entities), and external general entities referenced
104     * within the document element (if the parser reads external general
105     * entities). The application may request that the parser locate
106     * the entity itself, that it use an alternative URI, or that it
107     * use data provided by the application (as a character or byte
108     * input stream).</p>
109     *
110     * <p>Application writers can use this method to redirect external
111     * system identifiers to secure and/or local URIs, to look up
112     * public identifiers in a catalogue, or to read an entity from a
113     * database or other input source (including, for example, a dialog
114     * box). Neither XML nor SAX specifies a preferred policy for using
115     * public or system IDs to resolve resources. However, SAX specifies
116     * how to interpret any InputSource returned by this method, and that
117     * if none is returned, then the system ID will be dereferenced as
118     * a URL. </p>
119     *
120     * <p>If the system identifier is a URL, the SAX parser must
121     * resolve it fully before reporting it to the application.</p>
122     *
123     * @param publicId The public identifier of the external entity
124     *       being referenced, or null if none was supplied.
125     * @param systemId The system identifier of the external entity
126     *       being referenced.
127     * @return An InputSource object describing the new input source,
128     *       or null to request that the parser open a regular
129     *       URI connection to the system identifier.
130     * @exception org.xml.sax.SAXException Any SAX exception, possibly
131     *       wrapping another exception.
132     * @exception java.io.IOException A Java-specific IO exception,
133     *       possibly the result of creating a new InputStream
134     *       or Reader for the InputSource.
135     * @see org.xml.sax.InputSource
136     */
137     public abstract InputSource resolveEntity (String publicId,
138                                             String systemId)
139         throws SAXException, IOException;
140
141 }
142
143 // end of EntityResolver.java

```

## 47.7 ErrorHandler.java

Secuencia 484: org/xml/sax/ErrorHandler.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.

```

```
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX error handler.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: ErrorHandler.java,v 1.2 2004/11/03 22:44:52 jsuttor Exp $
30
31 package org.xml.sax;
32
33
34 /**
35  * Basic interface for SAX error handlers.
36  *
37  * <blockquote>
38  * <em>This module, both source code and documentation, is in the
39  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
40  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
41  * for further information.
42  * </blockquote>
43  *
44  * <p>If a SAX application needs to implement customized error
45  * handling, it must implement this interface and then register an
46  * instance with the XML reader using the
47  * {@link org.xml.sax.XMLReader#setErrorHandler setErrorHandler}
48  * method. The parser will then report all errors and warnings
49  * through this interface.</p>
50  *
51  * <p><strong>WARNING:</strong> If an application does <em>not</em>
52  * register an ErrorHandler, XML parsing errors will go unreported,
53  * except that <em>SAXParseException</em>s will be thrown for fatal errors.
54  * In order to detect validity errors, an ErrorHandler that does something
55  * with {@link #error error()} calls must be registered.</p>
```

```

56  *
57  * <p>For XML processing errors, a SAX driver must use this interface
58  * in preference to throwing an exception: it is up to the application
59  * to decide whether to throw an exception for different types of
60  * errors and warnings. Note, however, that there is no requirement that
61  * the parser continue to report additional errors after a call to
62  * {@link #fatalError fatalError}. In other words, a SAX driver class
63  * may throw an exception after reporting any fatalError.
64  * Also parsers may throw appropriate exceptions for non-XML errors.
65  * For example, {@link XMLReader#parse XMLReader.parse()} would throw
66  * an IOException for errors accessing entities or the document.</p>
67  *
68  * @since SAX 1.0
69  * @author David Megginson
70  * @see org.xml.sax.XMLReader#setErrorHandler
71  * @see org.xml.sax.SAXParseException
72  */
73  public interface ErrorHandler {
74
75      /**
76       * Receive notification of a warning.
77       *
78       * <p>SAX parsers will use this method to report conditions that
79       * are not errors or fatal errors as defined by the XML
80       * recommendation. The default behaviour is to take no
81       * action.</p>
82       *
83       * <p>The SAX parser must continue to provide normal parsing events
84       * after invoking this method: it should still be possible for the
85       * application to process the document through to the end.</p>
86       *
87       * <p>Filters may use this method to report other, non-XML warnings
88       * as well.</p>
89       *
90       * @param exception The warning information encapsulated in a
91       *                   SAX parse exception.
92       * @exception org.xml.sax.SAXException Any SAX exception, possibly
93       *                   wrapping another exception.
94       * @see org.xml.sax.SAXParseException
95       */
96
97      public abstract void warning (SAXParseException exception)
98          throws SAXException;
99
100
101      /**
102       * Receive notification of a recoverable error.
103       *
104       * <p>This corresponds to the definition of "error" in section 1.2
105       * of the W3C XML 1.0 Recommendation. For example, a validating
106       * parser would use this callback to report the violation of a
107       * validity constraint. The default behaviour is to take no
108       * action.</p>

```

```

109  *
110  * <p>The SAX parser must continue to provide normal parsing
111  * events after invoking this method: it should still be possible
112  * for the application to process the document through to the end.
113  * If the application cannot do so, then the parser should report
114  * a fatal error even if the XML recommendation does not require
115  * it to do so.</p>
116  *
117  * <p>Filters may use this method to report other, non-XML errors
118  * as well.</p>
119  *
120  * @param exception The error information encapsulated in a
121  *                  SAX parse exception.
122  * @exception org.xml.sax.SAXException Any SAX exception, possibly
123  *                  wrapping another exception.
124  * @see org.xml.sax.SAXParseException
125  */
126  public abstract void error (SAXParseException exception)
127      throws SAXException;
128
129
130  /**
131   * Receive notification of a non-recoverable error.
132   *
133   * <p><strong>There is an apparent contradiction between the
134   * documentation for this method and the documentation for {@link
135   * org.xml.sax.ContentHandler#endDocument}. Until this ambiguity
136   * is resolved in a future major release, clients should make no
137   * assumptions about whether endDocument() will or will not be
138   * invoked when the parser has reported a fatalError() or thrown
139   * an exception.</strong></p>
140   *
141   * <p>This corresponds to the definition of "fatal error" in
142   * section 1.2 of the W3C XML 1.0 Recommendation. For example, a
143   * parser would use this callback to report the violation of a
144   * well-formedness constraint.</p>
145   *
146   * <p>The application must assume that the document is unusable
147   * after the parser has invoked this method, and should continue
148   * (if at all) only for the sake of collecting additional error
149   * messages: in fact, SAX parsers are free to stop reporting any
150   * other events once this method has been invoked.</p>
151   *
152   * @param exception The error information encapsulated in a
153   *                  SAX parse exception.
154   * @exception org.xml.sax.SAXException Any SAX exception, possibly
155   *                  wrapping another exception.
156   * @see org.xml.sax.SAXParseException
157   */
158  public abstract void fatalError (SAXParseException exception)
159      throws SAXException;
160
161  }

```

```
162
163 // end of ErrorHandler.java
```

## 47.8 HandlerBase.java

Secuencia 485: org/xml/sax/HandlerBase.java

```
1  /*
2  * Copyright (c) 2000, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX default handler base class.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: HandlerBase.java,v 1.2 2005/06/10 03:50:47 jeffsuttor Exp $
30
31 package org.xml.sax;
32
33 /**
34  * Default base class for handlers.
35  *
36  * <blockquote>
37  * <em>This module, both source code and documentation, is in the
38  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
39  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
40  * for further information.
41  * </blockquote>
42  *
43  * <p>This class implements the default behaviour for four SAX1
44  * interfaces: EntityResolver, DTDHandler, DocumentHandler,
45  * and ErrorHandler. It is now obsolete, but is included in SAX2 to
46  * support legacy SAX1 applications. SAX2 applications should use
47  * the {@link org.xml.sax.helpers.DefaultHandler DefaultHandler}
```

```

48  * class instead.</p>
49  *
50  * <p>Application writers can extend this class when they need to
51  * implement only part of an interface; parser writers can
52  * instantiate this class to provide default handlers when the
53  * application has not supplied its own.</p>
54  *
55  * <p>Note that the use of this class is optional.</p>
56  *
57  * @deprecated This class works with the deprecated
58  *             {@link org.xml.sax.DocumentHandler DocumentHandler}
59  *             interface. It has been replaced by the SAX2
60  *             {@link org.xml.sax.helpers.DefaultHandler DefaultHandler}
61  *             class.
62  * @since SAX 1.0
63  * @author David Megginson
64  * @see org.xml.sax.EntityResolver
65  * @see org.xml.sax.DTDHandler
66  * @see org.xml.sax.DocumentHandler
67  * @see org.xml.sax.ErrorHandler
68  */
69  public class HandlerBase
70      implements EntityResolver, DTDHandler, DocumentHandler, ErrorHandler
71  {
72
73
74      ////////////////////////////////////////////////////
75      // Default implementation of the EntityResolver interface.
76      ////////////////////////////////////////////////////
77
78      /**
79       * Resolve an external entity.
80       *
81       * <p>Always return null, so that the parser will use the system
82       * identifier provided in the XML document. This method implements
83       * the SAX default behaviour: application writers can override it
84       * in a subclass to do special translations such as catalog lookups
85       * or URI redirection.</p>
86       *
87       * @param publicId The public identifier, or null if none is
88       *                 available.
89       * @param systemId The system identifier provided in the XML
90       *                 document.
91       * @return The new input source, or null to require the
92       *         default behaviour.
93       * @exception org.xml.sax.SAXException Any SAX exception, possibly
94       *         wrapping another exception.
95       * @see org.xml.sax.EntityResolver#resolveEntity
96       */
97      public InputSource resolveEntity (String publicId, String systemId)
98          throws SAXException
99      {
100          return null;

```



```

101     }
102
103
104
105     //////////////////////////////////////
106     // Default implementation of DTDHandler interface.
107     //////////////////////////////////////
108
109
110     /**
111      * Receive notification of a notation declaration.
112      *
113      * <p>By default, do nothing. Application writers may override this
114      * method in a subclass if they wish to keep track of the notations
115      * declared in a document.</p>
116      *
117      * @param name The notation name.
118      * @param publicId The notation public identifier, or null if not
119      *                 available.
120      * @param systemId The notation system identifier.
121      * @see org.xml.sax.DTDHandler#notationDecl
122      */
123     public void notationDecl (String name, String publicId, String systemId)
124     {
125         // no op
126     }
127
128
129     /**
130      * Receive notification of an unparsed entity declaration.
131      *
132      * <p>By default, do nothing. Application writers may override this
133      * method in a subclass to keep track of the unparsed entities
134      * declared in a document.</p>
135      *
136      * @param name The entity name.
137      * @param publicId The entity public identifier, or null if not
138      *                 available.
139      * @param systemId The entity system identifier.
140      * @param notationName The name of the associated notation.
141      * @see org.xml.sax.DTDHandler#unparsedEntityDecl
142      */
143     public void unparsedEntityDecl (String name, String publicId,
144                                     String systemId, String notationName)
145     {
146         // no op
147     }
148
149
150
151     //////////////////////////////////////
152     // Default implementation of DocumentHandler interface.
153     //////////////////////////////////////

```

```
154
155
156 /**
157  * Receive a Locator object for document events.
158  *
159  * <p>By default, do nothing. Application writers may override this
160  * method in a subclass if they wish to store the locator for use
161  * with other document events.</p>
162  *
163  * @param locator A locator for all SAX document events.
164  * @see org.xml.sax.DocumentHandler#setDocumentLocator
165  * @see org.xml.sax.Locator
166  */
167 public void setDocumentLocator (Locator locator)
168 {
169     // no op
170 }
171
172
173 /**
174  * Receive notification of the beginning of the document.
175  *
176  * <p>By default, do nothing. Application writers may override this
177  * method in a subclass to take specific actions at the beginning
178  * of a document (such as allocating the root node of a tree or
179  * creating an output file).</p>
180  *
181  * @exception org.xml.sax.SAXException Any SAX exception, possibly
182  *         wrapping another exception.
183  * @see org.xml.sax.DocumentHandler#startDocument
184  */
185 public void startDocument ()
186     throws SAXException
187 {
188     // no op
189 }
190
191
192 /**
193  * Receive notification of the end of the document.
194  *
195  * <p>By default, do nothing. Application writers may override this
196  * method in a subclass to take specific actions at the end
197  * of a document (such as finalising a tree or closing an output
198  * file).</p>
199  *
200  * @exception org.xml.sax.SAXException Any SAX exception, possibly
201  *         wrapping another exception.
202  * @see org.xml.sax.DocumentHandler#endDocument
203  */
204 public void endDocument ()
205     throws SAXException
206 {
```

```
207     // no op
208 }
209
210
211 /**
212  * Receive notification of the start of an element.
213  *
214  * <p>By default, do nothing. Application writers may override this
215  * method in a subclass to take specific actions at the start of
216  * each element (such as allocating a new tree node or writing
217  * output to a file).</p>
218  *
219  * @param name The element type name.
220  * @param attributes The specified or defaulted attributes.
221  * @exception org.xml.sax.SAXException Any SAX exception, possibly
222  *         wrapping another exception.
223  * @see org.xml.sax.DocumentHandler#startElement
224  */
225 public void startElement (String name, AttributeList attributes)
226     throws SAXException
227 {
228     // no op
229 }
230
231
232 /**
233  * Receive notification of the end of an element.
234  *
235  * <p>By default, do nothing. Application writers may override this
236  * method in a subclass to take specific actions at the end of
237  * each element (such as finalising a tree node or writing
238  * output to a file).</p>
239  *
240  * @param name the element name
241  * @exception org.xml.sax.SAXException Any SAX exception, possibly
242  *         wrapping another exception.
243  * @see org.xml.sax.DocumentHandler#endElement
244  */
245 public void endElement (String name)
246     throws SAXException
247 {
248     // no op
249 }
250
251
252 /**
253  * Receive notification of character data inside an element.
254  *
255  * <p>By default, do nothing. Application writers may override this
256  * method to take specific actions for each chunk of character data
257  * (such as adding the data to a node or buffer, or printing it to
258  * a file).</p>
259  *
```

```
260     * @param ch The characters.
261     * @param start The start position in the character array.
262     * @param length The number of characters to use from the
263     *         character array.
264     * @exception org.xml.sax.SAXException Any SAX exception, possibly
265     *         wrapping another exception.
266     * @see org.xml.sax.DocumentHandler#characters
267     */
268     public void characters (char ch[], int start, int length)
269         throws SAXException
270     {
271         // no op
272     }
273
274
275     /**
276     * Receive notification of ignorable whitespace in element content.
277     *
278     * <p>By default, do nothing. Application writers may override this
279     * method to take specific actions for each chunk of ignorable
280     * whitespace (such as adding data to a node or buffer, or printing
281     * it to a file).</p>
282     *
283     * @param ch The whitespace characters.
284     * @param start The start position in the character array.
285     * @param length The number of characters to use from the
286     *         character array.
287     * @exception org.xml.sax.SAXException Any SAX exception, possibly
288     *         wrapping another exception.
289     * @see org.xml.sax.DocumentHandler#ignorableWhitespace
290     */
291     public void ignorableWhitespace (char ch[], int start, int length)
292         throws SAXException
293     {
294         // no op
295     }
296
297
298     /**
299     * Receive notification of a processing instruction.
300     *
301     * <p>By default, do nothing. Application writers may override this
302     * method in a subclass to take specific actions for each
303     * processing instruction, such as setting status variables or
304     * invoking other methods.</p>
305     *
306     * @param target The processing instruction target.
307     * @param data The processing instruction data, or null if
308     *         none is supplied.
309     * @exception org.xml.sax.SAXException Any SAX exception, possibly
310     *         wrapping another exception.
311     * @see org.xml.sax.DocumentHandler#processingInstruction
312     */
```

```

313 public void processingInstruction (String target, String data)
314     throws SAXException
315 {
316     // no op
317 }
318
319
320
321 ///////////////////////////////////////////////////
322 // Default implementation of the ErrorHandler interface.
323 ///////////////////////////////////////////////////
324
325
326 /**
327  * Receive notification of a parser warning.
328  *
329  * <p>The default implementation does nothing. Application writers
330  * may override this method in a subclass to take specific actions
331  * for each warning, such as inserting the message in a log file or
332  * printing it to the console.</p>
333  *
334  * @param e The warning information encoded as an exception.
335  * @exception org.xml.sax.SAXException Any SAX exception, possibly
336  *         wrapping another exception.
337  * @see org.xml.sax.ErrorHandler#warning
338  * @see org.xml.sax.SAXParseException
339  */
340 public void warning (SAXParseException e)
341     throws SAXException
342 {
343     // no op
344 }
345
346
347 /**
348  * Receive notification of a recoverable parser error.
349  *
350  * <p>The default implementation does nothing. Application writers
351  * may override this method in a subclass to take specific actions
352  * for each error, such as inserting the message in a log file or
353  * printing it to the console.</p>
354  *
355  * @param e The warning information encoded as an exception.
356  * @exception org.xml.sax.SAXException Any SAX exception, possibly
357  *         wrapping another exception.
358  * @see org.xml.sax.ErrorHandler#warning
359  * @see org.xml.sax.SAXParseException
360  */
361 public void error (SAXParseException e)
362     throws SAXException
363 {
364     // no op
365 }

```

```

366
367
368 /**
369  * Report a fatal XML parsing error.
370  *
371  * <p>The default implementation throws a SAXParseException.
372  * Application writers may override this method in a subclass if
373  * they need to take specific actions for each fatal error (such as
374  * collecting all of the errors into a single report): in any case,
375  * the application must stop all regular processing when this
376  * method is invoked, since the document is no longer reliable, and
377  * the parser may no longer report parsing events.</p>
378  *
379  * @param e The error information encoded as an exception.
380  * @exception org.xml.sax.SAXException Any SAX exception, possibly
381  *         wrapping another exception.
382  * @see org.xml.sax.ErrorHandler#fatalError
383  * @see org.xml.sax.SAXParseException
384  */
385 public void fatalError (SAXParseException e)
386     throws SAXException
387 {
388     throw e;
389 }
390
391 }
392
393 // end of HandlerBase.java

```

## 47.9 InputSource.java

Secuencia 486: org/xml/sax/InputSource.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```
22  *
23  *
24  */
25
26  // SAX input source.
27  // http://www.saxproject.org
28  // No warranty; no copyright -- use this as you will.
29  // $Id: InputSource.java,v 1.2 2004/11/03 22:55:32 jsuttor Exp $
30
31  package org.xml.sax;
32
33  import java.io.Reader;
34  import java.io.InputStream;
35
36  /**
37   * A single input source for an XML entity.
38   *
39   * <blockquote>
40   * <em>This module, both source code and documentation, is in the
41   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
42   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
43   * for further information.
44   * </blockquote>
45   *
46   * <p>This class allows a SAX application to encapsulate information
47   * about an input source in a single object, which may include
48   * a public identifier, a system identifier, a byte stream (possibly
49   * with a specified encoding), and/or a character stream.</p>
50   *
51   * <p>There are two places that the application can deliver an
52   * input source to the parser: as the argument to the Parser.parse
53   * method, or as the return value of the EntityResolver.resolveEntity
54   * method.</p>
55   *
56   * <p>The SAX parser will use the InputSource object to determine how
57   * to read XML input. If there is a character stream available, the
58   * parser will read that stream directly, disregarding any text
59   * encoding declaration found in that stream.
60   * If there is no character stream, but there is
61   * a byte stream, the parser will use that byte stream, using the
62   * encoding specified in the InputSource or else (if no encoding is
63   * specified) autodetecting the character encoding using an algorithm
64   * such as the one in the XML specification. If neither a character
65   * stream nor a
66   * byte stream is available, the parser will attempt to open a URI
67   * connection to the resource identified by the system
68   * identifier.</p>
69   *
70   * <p>An InputSource object belongs to the application: the SAX parser
71   * shall never modify it in any way (it may modify a copy if
72   * necessary). However, standard processing of both byte and
73   * character streams is to close them on as part of end-of-parse cleanup,
74   * so applications should not attempt to re-use such streams after they
```

```
75  * have been handed to a parser.  </p>
76  *
77  * @since SAX 1.0
78  * @author David Megginson
79  * @see org.xml.sax.XMLReader#parse(org.xml.sax.InputSource)
80  * @see org.xml.sax.EntityResolver#resolveEntity
81  * @see java.io.InputStream
82  * @see java.io.Reader
83  */
84  public class InputSource {
85
86      /**
87       * Zero-argument default constructor.
88       *
89       * @see #setPublicId
90       * @see #setSystemId
91       * @see #setByteStream
92       * @see #setCharacterStream
93       * @see #setEncoding
94       */
95      public InputSource ()
96      {
97      }
98
99
100     /**
101      * Create a new input source with a system identifier.
102      *
103      * <p>Applications may use setPublicId to include a
104      * public identifier as well, or setEncoding to specify
105      * the character encoding, if known.</p>
106      *
107      * <p>If the system identifier is a URL, it must be fully
108      * resolved (it may not be a relative URL).</p>
109      *
110      * @param systemId The system identifier (URI).
111      * @see #setPublicId
112      * @see #setSystemId
113      * @see #setByteStream
114      * @see #setEncoding
115      * @see #setCharacterStream
116      */
117      public InputSource (String systemId)
118      {
119          setSystemId(systemId);
120      }
121
122
123     /**
124      * Create a new input source with a byte stream.
125      *
126      * <p>Application writers should use setSystemId() to provide a base
127      * for resolving relative URIs, may use setPublicId to include a
```



```
128     * public identifier, and may use setEncoding to specify the object's
129     * character encoding.</p>
130     *
131     * @param byteStream The raw byte stream containing the document.
132     * @see #setPublicId
133     * @see #setSystemId
134     * @see #setEncoding
135     * @see #setByteStream
136     * @see #setCharacterStream
137     */
138     public InputSource (InputStream byteStream)
139     {
140         setByteStream(byteStream);
141     }
142
143
144     /**
145     * Create a new input source with a character stream.
146     *
147     * <p>Application writers should use setSystemId() to provide a base
148     * for resolving relative URIs, and may use setPublicId to include a
149     * public identifier.</p>
150     *
151     * <p>The character stream shall not include a byte order mark.</p>
152     *
153     * @see #setPublicId
154     * @see #setSystemId
155     * @see #setByteStream
156     * @see #setCharacterStream
157     */
158     public InputSource (Reader characterStream)
159     {
160         setCharacterStream(characterStream);
161     }
162
163
164     /**
165     * Set the public identifier for this input source.
166     *
167     * <p>The public identifier is always optional: if the application
168     * writer includes one, it will be provided as part of the
169     * location information.</p>
170     *
171     * @param publicId The public identifier as a string.
172     * @see #getPublicId
173     * @see org.xml.sax.Locator#getPublicId
174     * @see org.xml.sax.SAXParseException#getPublicId
175     */
176     public void setPublicId (String publicId)
177     {
178         this.publicId = publicId;
179     }
180
```

```
181
182 /**
183  * Get the public identifier for this input source.
184  *
185  * @return The public identifier, or null if none was supplied.
186  * @see #setPublicId
187  */
188 public String getPublicId ()
189 {
190     return publicId;
191 }
192
193
194 /**
195  * Set the system identifier for this input source.
196  *
197  * <p>The system identifier is optional if there is a byte stream
198  * or a character stream, but it is still useful to provide one,
199  * since the application can use it to resolve relative URIs
200  * and can include it in error messages and warnings (the parser
201  * will attempt to open a connection to the URI only if
202  * there is no byte stream or character stream specified).</p>
203  *
204  * <p>If the application knows the character encoding of the
205  * object pointed to by the system identifier, it can register
206  * the encoding using the setEncoding method.</p>
207  *
208  * <p>If the system identifier is a URL, it must be fully
209  * resolved (it may not be a relative URL).</p>
210  *
211  * @param systemId The system identifier as a string.
212  * @see #setEncoding
213  * @see #getSystemId
214  * @see org.xml.sax.Locator#getSystemId
215  * @see org.xml.sax.SAXParseException#getSystemId
216  */
217 public void setSystemId (String systemId)
218 {
219     this.systemId = systemId;
220 }
221
222
223 /**
224  * Get the system identifier for this input source.
225  *
226  * <p>The getEncoding method will return the character encoding
227  * of the object pointed to, or null if unknown.</p>
228  *
229  * <p>If the system ID is a URL, it will be fully resolved.</p>
230  *
231  * @return The system identifier, or null if none was supplied.
232  * @see #setSystemId
233  * @see #getEncoding
```

```
234     */
235     public String getSystemId ()
236     {
237         return systemId;
238     }
239
240
241     /**
242     * Set the byte stream for this input source.
243     *
244     * <p>The SAX parser will ignore this if there is also a character
245     * stream specified, but it will use a byte stream in preference
246     * to opening a URI connection itself.</p>
247     *
248     * <p>If the application knows the character encoding of the
249     * byte stream, it should set it with the setEncoding method.</p>
250     *
251     * @param byteStream A byte stream containing an XML document or
252     *     other entity.
253     * @see #setEncoding
254     * @see #getByteStream
255     * @see #getEncoding
256     * @see java.io.InputStream
257     */
258     public void setByteStream (InputStream byteStream)
259     {
260         this.byteStream = byteStream;
261     }
262
263
264     /**
265     * Get the byte stream for this input source.
266     *
267     * <p>The getEncoding method will return the character
268     * encoding for this byte stream, or null if unknown.</p>
269     *
270     * @return The byte stream, or null if none was supplied.
271     * @see #getEncoding
272     * @see #setByteStream
273     */
274     public InputStream getByteStream ()
275     {
276         return byteStream;
277     }
278
279
280     /**
281     * Set the character encoding, if known.
282     *
283     * <p>The encoding must be a string acceptable for an
284     * XML encoding declaration (see section 4.3.3 of the XML 1.0
285     * recommendation).</p>
286     *
```

```
287      * <p>This method has no effect when the application provides a
288      * character stream.</p>
289      *
290      * @param encoding A string describing the character encoding.
291      * @see #setSystemId
292      * @see #setByteStream
293      * @see #getEncoding
294      */
295      public void setEncoding (String encoding)
296      {
297          this.encoding = encoding;
298      }
299
300
301      /**
302      * Get the character encoding for a byte stream or URI.
303      * This value will be ignored when the application provides a
304      * character stream.
305      *
306      * @return The encoding, or null if none was supplied.
307      * @see #setByteStream
308      * @see #getSystemId
309      * @see #getByteStream
310      */
311      public String getEncoding ()
312      {
313          return encoding;
314      }
315
316
317      /**
318      * Set the character stream for this input source.
319      *
320      * <p>If there is a character stream specified, the SAX parser
321      * will ignore any byte stream and will not attempt to open
322      * a URI connection to the system identifier.</p>
323      *
324      * @param characterStream The character stream containing the
325      *       XML document or other entity.
326      * @see #getCharacterStream
327      * @see java.io.Reader
328      */
329      public void setCharacterStream (Reader characterStream)
330      {
331          this.characterStream = characterStream;
332      }
333
334
335      /**
336      * Get the character stream for this input source.
337      *
338      * @return The character stream, or null if none was supplied.
339      * @see #setCharacterStream
```

```

340     */
341     public Reader getCharacterStream ()
342     {
343         return characterStream;
344     }
345
346
347
348     //////////////////////////////////////
349     // Internal state.
350     //////////////////////////////////////
351
352     private String publicId;
353     private String systemId;
354     private InputStream byteStream;
355     private String encoding;
356     private Reader characterStream;
357
358 }
359
360 // end of InputSource.java

```

## 47.10 Locator.java

Secuencia 487: org/xml/sax/Locator.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX locator interface for document events.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.

```

```
29 // $Id: Locator.java,v 1.2 2004/11/03 22:55:32 jsuttor Exp $
30
31 package org.xml.sax;
32
33
34 /**
35  * Interface for associating a SAX event with a document location.
36  *
37  * <blockquote>
38  * <em>This module, both source code and documentation, is in the
39  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
40  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
41  * for further information.
42  * </blockquote>
43  *
44  * <p>If a SAX parser provides location information to the SAX
45  * application, it does so by implementing this interface and then
46  * passing an instance to the application using the content
47  * handler's {@link org.xml.sax.ContentHandler#setDocumentLocator
48  * setDocumentLocator} method. The application can use the
49  * object to obtain the location of any other SAX event
50  * in the XML source document.</p>
51  *
52  * <p>Note that the results returned by the object will be valid only
53  * during the scope of each callback method: the application
54  * will receive unpredictable results if it attempts to use the
55  * locator at any other time, or after parsing completes.</p>
56  *
57  * <p>SAX parsers are not required to supply a locator, but they are
58  * very strongly encouraged to do so. If the parser supplies a
59  * locator, it must do so before reporting any other document events.
60  * If no locator has been set by the time the application receives
61  * the {@link org.xml.sax.ContentHandler#startDocument startDocument}
62  * event, the application should assume that a locator is not
63  * available.</p>
64  *
65  * @since SAX 1.0
66  * @author David Megginson
67  * @see org.xml.sax.ContentHandler#setDocumentLocator
68  */
69 public interface Locator {
70
71
72     /**
73      * Return the public identifier for the current document event.
74      *
75      * <p>The return value is the public identifier of the document
76      * entity or of the external parsed entity in which the markup
77      * triggering the event appears.</p>
78      *
79      * @return A string containing the public identifier, or
80      *         null if none is available.
81      * @see #getSystemId
```

```

82      */
83      public abstract String getPublicId ();
84
85
86      /**
87       * Return the system identifier for the current document event.
88       *
89       * <p>The return value is the system identifier of the document
90       * entity or of the external parsed entity in which the markup
91       * triggering the event appears.</p>
92       *
93       * <p>If the system identifier is a URL, the parser must resolve it
94       * fully before passing it to the application. For example, a file
95       * name must always be provided as a <em>file:...</em> URL, and other
96       * kinds of relative URI are also resolved against their bases.</p>
97       *
98       * @return A string containing the system identifier, or null
99       *         if none is available.
100      * @see #getPublicId
101      */
102      public abstract String getSystemId ();
103
104
105      /**
106       * Return the line number where the current document event ends.
107       * Lines are delimited by line ends, which are defined in
108       * the XML specification.
109       *
110       * <p><strong>Warning:</strong> The return value from the method
111       * is intended only as an approximation for the sake of diagnostics;
112       * it is not intended to provide sufficient information
113       * to edit the character content of the original XML document.
114       * In some cases, these "line" numbers match what would be displayed
115       * as columns, and in others they may not match the source text
116       * due to internal entity expansion. </p>
117       *
118       * <p>The return value is an approximation of the line number
119       * in the document entity or external parsed entity where the
120       * markup triggering the event appears.</p>
121       *
122       * <p>If possible, the SAX driver should provide the line position
123       * of the first character after the text associated with the document
124       * event. The first line is line 1.</p>
125       *
126       * @return The line number, or -1 if none is available.
127       * @see #getColumnNumber
128       */
129      public abstract int getLineNumber ();
130
131
132      /**
133       * Return the column number where the current document event ends.
134       * This is one-based number of Java <code>char</code> values since

```

```

135     * the last line end.
136     *
137     * <p><strong>Warning:</strong> The return value from the method
138     * is intended only as an approximation for the sake of diagnostics;
139     * it is not intended to provide sufficient information
140     * to edit the character content of the original XML document.
141     * For example, when lines contain combining character sequences, wide
142     * characters, surrogate pairs, or bi-directional text, the value may
143     * not correspond to the column in a text editor's display. </p>
144     *
145     * <p>The return value is an approximation of the column number
146     * in the document entity or external parsed entity where the
147     * markup triggering the event appears.</p>
148     *
149     * <p>If possible, the SAX driver should provide the line position
150     * of the first character after the text associated with the document
151     * event. The first column in each line is column 1.</p>
152     *
153     * @return The column number, or -1 if none is available.
154     * @see #getLineNumber
155     */
156     public abstract int getColumnNumber ();
157
158 }
159
160 // end of Locator.java

```

## 47.11 Parser.java

Secuencia 488: org/xml/sax/Parser.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *

```



```
24  */
25
26  // SAX parser interface.
27  // http://www.saxproject.org
28  // No warranty; no copyright -- use this as you will.
29  // $Id: Parser.java,v 1.2 2004/11/03 22:55:32 jsuttor Exp $
30
31  package org.xml.sax;
32
33  import java.io.IOException;
34  import java.util.Locale;
35
36
37  /**
38   * Basic interface for SAX (Simple API for XML) parsers.
39   *
40   * <blockquote>
41   * <em>This module, both source code and documentation, is in the
42   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
43   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
44   * for further information.
45   * </blockquote>
46   *
47   * <p>This was the main event supplier interface for SAX1; it has
48   * been replaced in SAX2 by {@link org.xml.sax.XMLReader XMLReader},
49   * which includes Namespace support and sophisticated configurability
50   * and extensibility.</p>
51   *
52   * <p>All SAX1 parsers must implement this basic interface: it allows
53   * applications to register handlers for different types of events
54   * and to initiate a parse from a URI, or a character stream.</p>
55   *
56   * <p>All SAX1 parsers must also implement a zero-argument constructor
57   * (though other constructors are also allowed).</p>
58   *
59   * <p>SAX1 parsers are reusable but not re-entrant: the application
60   * may reuse a parser object (possibly with a different input source)
61   * once the first parse has completed successfully, but it may not
62   * invoke the parse() methods recursively within a parse.</p>
63   *
64   * @deprecated This interface has been replaced by the SAX2
65   *      {@link org.xml.sax.XMLReader XMLReader}
66   *      interface, which includes Namespace support.
67   * @since SAX 1.0
68   * @author David Megginson
69   * @see org.xml.sax.EntityResolver
70   * @see org.xml.sax.DTDHandler
71   * @see org.xml.sax.DocumentHandler
72   * @see org.xml.sax.ErrorHandler
73   * @see org.xml.sax.HandlerBase
74   * @see org.xml.sax.InputSource
75   */
76  public interface Parser
```

```
77 {
78
79 /**
80  * Allow an application to request a locale for errors and warnings.
81  *
82  * <p>SAX parsers are not required to provide localisation for errors
83  * and warnings; if they cannot support the requested locale,
84  * however, they must throw a SAX exception. Applications may
85  * not request a locale change in the middle of a parse.</p>
86  *
87  * @param locale A Java Locale object.
88  * @exception org.xml.sax.SAXException Throws an exception
89  *         (using the previous or default locale) if the
90  *         requested locale is not supported.
91  * @see org.xml.sax.SAXException
92  * @see org.xml.sax.SAXParseException
93  */
94 public abstract void setLocale (Locale locale)
95     throws SAXException;
96
97
98 /**
99  * Allow an application to register a custom entity resolver.
100  *
101  * <p>If the application does not register an entity resolver, the
102  * SAX parser will resolve system identifiers and open connections
103  * to entities itself (this is the default behaviour implemented in
104  * HandlerBase).</p>
105  *
106  * <p>Applications may register a new or different entity resolver
107  * in the middle of a parse, and the SAX parser must begin using
108  * the new resolver immediately.</p>
109  *
110  * @param resolver The object for resolving entities.
111  * @see EntityResolver
112  * @see HandlerBase
113  */
114 public abstract void setEntityResolver (EntityResolver resolver);
115
116
117 /**
118  * Allow an application to register a DTD event handler.
119  *
120  * <p>If the application does not register a DTD handler, all DTD
121  * events reported by the SAX parser will be silently
122  * ignored (this is the default behaviour implemented by
123  * HandlerBase).</p>
124  *
125  * <p>Applications may register a new or different
126  * handler in the middle of a parse, and the SAX parser must
127  * begin using the new handler immediately.</p>
128  *
129  * @param handler The DTD handler.
```

```
130     * @see DTDHandler
131     * @see HandlerBase
132     */
133     public abstract void setDTDHandler (DTDHandler handler);
134
135
136     /**
137     * Allow an application to register a document event handler.
138     *
139     * <p>If the application does not register a document handler, all
140     * document events reported by the SAX parser will be silently
141     * ignored (this is the default behaviour implemented by
142     * HandlerBase).</p>
143     *
144     * <p>Applications may register a new or different handler in the
145     * middle of a parse, and the SAX parser must begin using the new
146     * handler immediately.</p>
147     *
148     * @param handler The document handler.
149     * @see DocumentHandler
150     * @see HandlerBase
151     */
152     public abstract void setDocumentHandler (DocumentHandler handler);
153
154
155     /**
156     * Allow an application to register an error event handler.
157     *
158     * <p>If the application does not register an error event handler,
159     * all error events reported by the SAX parser will be silently
160     * ignored, except for fatalError, which will throw a SAXException
161     * (this is the default behaviour implemented by HandlerBase).</p>
162     *
163     * <p>Applications may register a new or different handler in the
164     * middle of a parse, and the SAX parser must begin using the new
165     * handler immediately.</p>
166     *
167     * @param handler The error handler.
168     * @see ErrorHandler
169     * @see SAXException
170     * @see HandlerBase
171     */
172     public abstract void setErrorHandler (ErrorHandler handler);
173
174
175     /**
176     * Parse an XML document.
177     *
178     * <p>The application can use this method to instruct the SAX parser
179     * to begin parsing an XML document from any valid input
180     * source (a character stream, a byte stream, or a URI).</p>
181     *
182     * <p>Applications may not invoke this method while a parse is in
```

```

183 * progress (they should create a new Parser instead for each
184 * additional XML document). Once a parse is complete, an
185 * application may reuse the same Parser object, possibly with a
186 * different input source.</p>
187 *
188 * @param source The input source for the top-level of the
189 *       XML document.
190 * @exception org.xml.sax.SAXException Any SAX exception, possibly
191 *       wrapping another exception.
192 * @exception java.io.IOException An IO exception from the parser,
193 *       possibly from a byte stream or character stream
194 *       supplied by the application.
195 * @see org.xml.sax.InputSource
196 * @see #parse(java.lang.String)
197 * @see #setEntityResolver
198 * @see #setDTDHandler
199 * @see #setDocumentHandler
200 * @see #setErrorHandler
201 */
202 public abstract void parse (InputSource source)
203     throws SAXException, IOException;
204
205
206 /**
207 * Parse an XML document from a system identifier (URI).
208 *
209 * <p>This method is a shortcut for the common case of reading a
210 * document from a system identifier. It is the exact
211 * equivalent of the following:</p>
212 *
213 * <pre>
214 * parse(new InputSource(systemId));
215 * </pre>
216 *
217 * <p>If the system identifier is a URL, it must be fully resolved
218 * by the application before it is passed to the parser.</p>
219 *
220 * @param systemId The system identifier (URI).
221 * @exception org.xml.sax.SAXException Any SAX exception, possibly
222 *       wrapping another exception.
223 * @exception java.io.IOException An IO exception from the parser,
224 *       possibly from a byte stream or character stream
225 *       supplied by the application.
226 * @see #parse(org.xml.sax.InputSource)
227 */
228 public abstract void parse (String systemId)
229     throws SAXException, IOException;
230
231 }
232
233 // end of Parser.java

```

**47.12 SAXException.java**

Secuencia 489: org/xml/sax/SAXException.java

```
1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX exception class.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: SAXException.java,v 1.3 2004/11/03 22:55:32 jsuttor Exp $
30
31 package org.xml.sax;
32
33 /**
34 * Encapsulate a general SAX error or warning.
35 *
36 * <blockquote>
37 * <em>This module, both source code and documentation, is in the
38 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
39 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
40 * for further information.
41 * </blockquote>
42 *
43 * <p>This class can contain basic error or warning information from
44 * either the XML parser or the application: a parser writer or
45 * application writer can subclass it to provide additional
46 * functionality. SAX handlers may throw this exception or
47 * any exception subclassed from it.</p>
48 *
49 * <p>If the application needs to pass through other types of
50 * exceptions, it must wrap those exceptions in a SAXException
```

```
51 * or an exception derived from a SAXException.</p>
52 *
53 * <p>If the parser or application needs to include information about a
54 * specific location in an XML document, it should use the
55 * {@link org.xml.sax.SAXParseException SAXParseException} subclass.</p>
56 *
57 * @since SAX 1.0
58 * @author David Megginson
59 * @version 2.0.1 (sax2r2)
60 * @see org.xml.sax.SAXParseException
61 */
62 public class SAXException extends Exception {
63
64
65     /**
66      * Create a new SAXException.
67      */
68     public SAXException ()
69     {
70         super();
71         this.exception = null;
72     }
73
74
75     /**
76      * Create a new SAXException.
77      *
78      * @param message The error or warning message.
79      */
80     public SAXException (String message) {
81         super(message);
82         this.exception = null;
83     }
84
85
86     /**
87      * Create a new SAXException wrapping an existing exception.
88      *
89      * <p>The existing exception will be embedded in the new
90      * one, and its message will become the default message for
91      * the SAXException.</p>
92      *
93      * @param e The exception to be wrapped in a SAXException.
94      */
95     public SAXException (Exception e)
96     {
97         super();
98         this.exception = e;
99     }
100
101
102     /**
103      * Create a new SAXException from an existing exception.
```

```
104      *
105      * <p>The existing exception will be embedded in the new
106      * one, but the new exception will have its own message.</p>
107      *
108      * @param message The detail message.
109      * @param e The exception to be wrapped in a SAXException.
110      */
111     public SAXException (String message, Exception e)
112     {
113         super(message);
114         this.exception = e;
115     }
116
117
118     /**
119     * Return a detail message for this exception.
120     *
121     * <p>If there is an embedded exception, and if the SAXException
122     * has no detail message of its own, this method will return
123     * the detail message from the embedded exception.</p>
124     *
125     * @return The error or warning message.
126     */
127     public String getMessage ()
128     {
129         String message = super.getMessage();
130
131         if (message == null && exception != null) {
132             return exception.getMessage();
133         } else {
134             return message;
135         }
136     }
137
138
139     /**
140     * Return the embedded exception, if any.
141     *
142     * @return The embedded exception, or null if there is none.
143     */
144     public Exception getException ()
145     {
146         return exception;
147     }
148
149     /**
150     * Return the cause of the exception
151     *
152     * @return Return the cause of the exception
153     */
154     public Throwable getCause() {
155         return exception;
156     }
```

```

157
158  /**
159   * Override toString to pick up any embedded exception.
160   *
161   * @return A string representation of this exception.
162   */
163  public String toString ()
164  {
165      if (exception != null) {
166          return super.toString() + "\n" + exception.toString();
167      } else {
168          return super.toString();
169      }
170  }
171
172
173
174  //////////////////////////////////////
175  // Internal state.
176  //////////////////////////////////////
177
178
179  /**
180   * @serial The embedded exception if tunnelling, or null.
181   */
182  private Exception exception;
183
184  // Added serialVersionUID to preserve binary compatibility
185  static final long serialVersionUID = 583241635256073760L;
186 }
187
188 // end of SAXException.java

```

### 47.13 SAXNotRecognizedException.java

Secuencia 490: org/xml/sax/SAXNotRecognizedException.java

```

1  /*
2   * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3   * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4   *
5   *
6   *
7   *
8   *
9   *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
17  *

```



```
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  // SAXNotRecognizedException.java - unrecognized feature or value.
27  // http://www.saxproject.org
28  // Written by David Megginson
29  // NO WARRANTY! This class is in the Public Domain.
30  // $Id: SAXNotRecognizedException.java,v 1.3 2004/11/03 22:55:32 jsuttor Exp $
31
32  package org.xml.sax;
33
34
35  /**
36   * Exception class for an unrecognized identifier.
37   *
38   * <blockquote>
39   * <em>This module, both source code and documentation, is in the
40   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
41   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
42   * for further information.
43   * </blockquote>
44   *
45   * <p>An XMLReader will throw this exception when it finds an
46   * unrecognized feature or property identifier; SAX applications and
47   * extensions may use this class for other, similar purposes.</p>
48   *
49   * @since SAX 2.0
50   * @author David Megginson
51   * @see org.xml.sax.SAXNotSupportedException
52   */
53  public class SAXNotRecognizedException extends SAXException
54  {
55
56      /**
57       * Default constructor.
58       */
59      public SAXNotRecognizedException ()
60      {
61          super();
62      }
63
64
65      /**
66       * Construct a new exception with the given message.
67       *
68       * @param message The text message of the exception.
69       */
70      public SAXNotRecognizedException (String message)
```

```

71     {
72         super(message);
73     }
74
75     // Added serialVersionUID to preserve binary compatibility
76     static final long serialVersionUID = 5440506620509557213L;
77 }
78
79 // end of SAXNotRecognizedException.java

```

#### 47.14 SAXNotSupportedException.java

Secuencia 491: org/xml/sax/SAXNotSupportedException.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAXNotSupportedException.java - unsupported feature or value.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the Public Domain.
30 // $Id: SAXNotSupportedException.java,v 1.4 2004/11/03 22:55:32 jsuttor Exp $
31
32 package org.xml.sax;
33
34 /**
35  * Exception class for an unsupported operation.
36  *
37  * <blockquote>
38  * <em>This module, both source code and documentation, is in the
39  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
40  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>

```

```

41  * for further information.
42  * </blockquote>
43  *
44  * <p>An XMLReader will throw this exception when it recognizes a
45  * feature or property identifier, but cannot perform the requested
46  * operation (setting a state or value). Other SAX2 applications and
47  * extensions may use this class for similar purposes.</p>
48  *
49  * @since SAX 2.0
50  * @author David Megginson
51  * @see org.xml.sax.SAXNotRecognizedException
52  */
53  public class SAXNotSupportedException extends SAXException
54  {
55
56      /**
57       * Construct a new exception with no message.
58       */
59      public SAXNotSupportedException ()
60      {
61          super();
62      }
63
64
65      /**
66       * Construct a new exception with the given message.
67       *
68       * @param message The text message of the exception.
69       */
70      public SAXNotSupportedException (String message)
71      {
72          super(message);
73      }
74
75      // Added serialVersionUID to preserve binary compatibility
76      static final long serialVersionUID = -1422818934641823846L;
77  }
78
79  // end of SAXNotSupportedException.java

```

## 47.15 SAXParseException.java

Secuencia 492: org/xml/sax/SAXParseException.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *

```

```

11  *
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  // SAX exception class.
27  // http://www.saxproject.org
28  // No warranty; no copyright -- use this as you will.
29  // $Id: SAXParseException.java,v 1.2 2004/11/03 22:55:32 jsuttor Exp $
30
31  package org.xml.sax;
32
33  /**
34   * Encapsulate an XML parse error or warning.
35   *
36   * <blockquote>
37   * <em>This module, both source code and documentation, is in the
38   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
39   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
40   * for further information.
41   * </blockquote>
42   *
43   * <p>This exception may include information for locating the error
44   * in the original XML document, as if it came from a {@link Locator}
45   * object. Note that although the application
46   * will receive a SAXParseException as the argument to the handlers
47   * in the {@link org.xml.sax.ErrorHandler ErrorHandler} interface,
48   * the application is not actually required to throw the exception;
49   * instead, it can simply read the information in it and take a
50   * different action.</p>
51   *
52   * <p>Since this exception is a subclass of {@link org.xml.sax.SAXException
53   * SAXException}, it inherits the ability to wrap another exception.</p>
54   *
55   * @since SAX 1.0
56   * @author David Megginson
57   * @version 2.0.1 (sax2r2)
58   * @see org.xml.sax.SAXException
59   * @see org.xml.sax.Locator
60   * @see org.xml.sax.ErrorHandler
61   */
62  public class SAXParseException extends SAXException {
63

```

```

64
65 ////////////////////////////////////////////////////
66 // Constructors.
67 ////////////////////////////////////////////////////
68
69
70 /**
71  * Create a new SAXParseException from a message and a Locator.
72  *
73  * <p>This constructor is especially useful when an application is
74  * creating its own exception from within a {@link org.xml.sax.ContentHandler
75  * ContentHandler} callback.</p>
76  *
77  * @param message The error or warning message.
78  * @param locator The locator object for the error or warning (may be
79  *      null).
80  * @see org.xml.sax.Locator
81  */
82 public SAXParseException (String message, Locator locator) {
83     super(message);
84     if (locator != null) {
85         init(locator.getPublicId(), locator.getSystemId(),
86             locator.getLineNumber(), locator.getColumnNumber());
87     } else {
88         init(null, null, -1, -1);
89     }
90 }
91
92
93 /**
94  * Wrap an existing exception in a SAXParseException.
95  *
96  * <p>This constructor is especially useful when an application is
97  * creating its own exception from within a {@link org.xml.sax.ContentHandler
98  * ContentHandler} callback, and needs to wrap an existing exception that is not a
99  * subclass of {@link org.xml.sax.SAXException SAXException}.</p>
100  *
101  * @param message The error or warning message, or null to
102  *      use the message from the embedded exception.
103  * @param locator The locator object for the error or warning (may be
104  *      null).
105  * @param e Any exception.
106  * @see org.xml.sax.Locator
107  */
108 public SAXParseException (String message, Locator locator,
109     Exception e) {
110     super(message, e);
111     if (locator != null) {
112         init(locator.getPublicId(), locator.getSystemId(),
113             locator.getLineNumber(), locator.getColumnNumber());
114     } else {
115         init(null, null, -1, -1);
116     }

```

```
117     }
118
119
120     /**
121      * Create a new SAXParseException.
122      *
123      * <p>This constructor is most useful for parser writers.</p>
124      *
125      * <p>All parameters except the message are as if
126      * they were provided by a {@link Locator}. For example, if the
127      * system identifier is a URL (including relative filename), the
128      * caller must resolve it fully before creating the exception.</p>
129      *
130      *
131      * @param message The error or warning message.
132      * @param publicId The public identifier of the entity that generated
133      *                  the error or warning.
134      * @param systemId The system identifier of the entity that generated
135      *                  the error or warning.
136      * @param lineNumber The line number of the end of the text that
137      *                    caused the error or warning.
138      * @param columnNumber The column number of the end of the text that
139      *                      cause the error or warning.
140      */
141     public SAXParseException (String message, String publicId, String systemId,
142                              int lineNumber, int columnNumber)
143     {
144         super(message);
145         init(publicId, systemId, lineNumber, columnNumber);
146     }
147
148
149     /**
150      * Create a new SAXParseException with an embedded exception.
151      *
152      * <p>This constructor is most useful for parser writers who
153      * need to wrap an exception that is not a subclass of
154      * {@link org.xml.sax.SAXException SAXException}</p>
155      *
156      * <p>All parameters except the message and exception are as if
157      * they were provided by a {@link Locator}. For example, if the
158      * system identifier is a URL (including relative filename), the
159      * caller must resolve it fully before creating the exception.</p>
160      *
161      * @param message The error or warning message, or null to use
162      *                  the message from the embedded exception.
163      * @param publicId The public identifier of the entity that generated
164      *                  the error or warning.
165      * @param systemId The system identifier of the entity that generated
166      *                  the error or warning.
167      * @param lineNumber The line number of the end of the text that
168      *                    caused the error or warning.
169      * @param columnNumber The column number of the end of the text that
```

```
170      *          cause the error or warning.
171      * @param e Another exception to embed in this one.
172      */
173      public SAXParseException (String message, String publicId, String systemId,
174                               int lineNumber, int columnNumber, Exception e)
175      {
176          super(message, e);
177          init(publicId, systemId, lineNumber, columnNumber);
178      }
179
180
181      /**
182       * Internal initialization method.
183       *
184       * @param publicId The public identifier of the entity which generated the exception,
185       *                or null.
186       * @param systemId The system identifier of the entity which generated the exception,
187       *                or null.
188       * @param lineNumber The line number of the error, or -1.
189       * @param columnNumber The column number of the error, or -1.
190       */
191      private void init (String publicId, String systemId,
192                       int lineNumber, int columnNumber)
193      {
194          this.publicId = publicId;
195          this.systemId = systemId;
196          this.lineNumber = lineNumber;
197          this.columnNumber = columnNumber;
198      }
199
200
201      /**
202       * Get the public identifier of the entity where the exception occurred.
203       *
204       * @return A string containing the public identifier, or null
205       *         if none is available.
206       * @see org.xml.sax.Locator#getPublicId
207       */
208      public String getPublicId ()
209      {
210          return this.publicId;
211      }
212
213
214      /**
215       * Get the system identifier of the entity where the exception occurred.
216       *
217       * <p>If the system identifier is a URL, it will have been resolved
218       * fully.</p>
219       *
220       * @return A string containing the system identifier, or null
221       *         if none is available.
222       * @see org.xml.sax.Locator#getSystemId
```

```
223     */
224     public String getSystemId ()
225     {
226         return this.systemId;
227     }
228
229
230     /**
231     * The line number of the end of the text where the exception occurred.
232     *
233     * <p>The first line is line 1.</p>
234     *
235     * @return An integer representing the line number, or -1
236     *         if none is available.
237     * @see org.xml.sax.Locator#getLineNumber
238     */
239     public int getLineNumber ()
240     {
241         return this.lineNumber;
242     }
243
244
245     /**
246     * The column number of the end of the text where the exception occurred.
247     *
248     * <p>The first column in a line is position 1.</p>
249     *
250     * @return An integer representing the column number, or -1
251     *         if none is available.
252     * @see org.xml.sax.Locator#getColumnNumber
253     */
254     public int getColumnNumber ()
255     {
256         return this.columnNumber;
257     }
258
259     /**
260     * Override toString to provide more detailed error message.
261     *
262     * @return A string representation of this exception.
263     */
264     public String toString() {
265         StringBuilder buf = new StringBuilder(getClass().getName());
266         String message = getLocalizedMessage();
267         if (publicId!=null)    buf.append("publicId: ").append(publicId);
268         if (systemId!=null)   buf.append("; systemId: ").append(systemId);
269         if (lineNumber!=-1)   buf.append("; lineNumber: ").append(lineNumber);
270         if (columnNumber!=-1) buf.append("; columnNumber: ").append(columnNumber);
271
272         //append the exception message at the end
273         if (message!=null)    buf.append("; ").append(message);
274         return buf.toString();
275     }
```



```

276
277 // Internal state.
278
279
280
281
282 /**
283  * @serial The public identifier, or null.
284  * @see #getPublicId
285  */
286 private String publicId;
287
288
289 /**
290  * @serial The system identifier, or null.
291  * @see #getSystemId
292  */
293 private String systemId;
294
295
296 /**
297  * @serial The line number, or -1.
298  * @see #getLineNumber
299  */
300 private int lineNumber;
301
302
303 /**
304  * @serial The column number, or -1.
305  * @see #getColumnNumber
306  */
307 private int columnNumber;
308
309 // Added serialVersionUID to preserve binary compatibility
310 static final long serialVersionUID = -5651165872476709336L;
311 }
312
313 // end of SAXParseException.java

```

## 47.16 XMLFilter.java

Secuencia 493: org/xml/sax/XMLFilter.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *

```

```
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26 // XMLFilter.java - filter SAX2 events.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the Public Domain.
30 // $Id: XMLFilter.java,v 1.2 2004/11/03 22:55:32 jsuttor Exp $
31
32 package org.xml.sax;
33
34
35 /**
36  * Interface for an XML filter.
37  *
38  * <blockquote>
39  * <em>This module, both source code and documentation, is in the
40  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
41  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
42  * for further information.
43  * </blockquote>
44  *
45  * <p>An XML filter is like an XML reader, except that it obtains its
46  * events from another XML reader rather than a primary source like
47  * an XML document or database. Filters can modify a stream of
48  * events as they pass on to the final application.</p>
49  *
50  * <p>The XMLFilterImpl helper class provides a convenient base
51  * for creating SAX2 filters, by passing on all {@link org.xml.sax.EntityResolver
52  * EntityResolver}, {@link org.xml.sax.DTDHandler DTDHandler},
53  * {@link org.xml.sax.ContentHandler ContentHandler} and {@link org.xml.sax.ErrorHandler
54  * ErrorHandler} events automatically.</p>
55  *
56  * @since SAX 2.0
57  * @author David Megginson
58  * @see org.xml.sax.helpers.XMLFilterImpl
59  */
60 public interface XMLFilter extends XMLReader
61 {
62
63     /**
64      * Set the parent reader.
```

```

65      *
66      * <p>This method allows the application to link the filter to
67      * a parent reader (which may be another filter). The argument
68      * may not be null.</p>
69      *
70      * @param parent The parent reader.
71      */
72      public abstract void setParent (XMLReader parent);
73
74
75      /**
76      * Get the parent reader.
77      *
78      * <p>This method allows the application to query the parent
79      * reader (which may be another filter). It is generally a
80      * bad idea to perform any operations on the parent reader
81      * directly: they should all pass through this filter.</p>
82      *
83      * @return The parent filter, or null if none has been set.
84      */
85      public abstract XMLReader getParent ();
86
87  }
88
89  // end of XMLFilter.java

```

## 47.17 XMLReader.java

Secuencia 494: org/xml/sax/XMLReader.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */

```

```
25
26 // XMLReader.java - read an XML document.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the Public Domain.
30 // $Id: XMLReader.java,v 1.3 2004/11/03 22:55:32 jsuttor Exp $
31
32 package org.xml.sax;
33
34 import java.io.IOException;
35
36
37 /**
38  * Interface for reading an XML document using callbacks.
39  *
40  * <blockquote>
41  * <em>This module, both source code and documentation, is in the
42  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
43  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
44  * for further information.
45  * </blockquote>
46  *
47  * <p><strong>Note:</strong> despite its name, this interface does
48  * <em>not</em> extend the standard Java {@link java.io.Reader Reader}
49  * interface, because reading XML is a fundamentally different activity
50  * than reading character data.</p>
51  *
52  * <p>XMLReader is the interface that an XML parser's SAX2 driver must
53  * implement. This interface allows an application to set and
54  * query features and properties in the parser, to register
55  * event handlers for document processing, and to initiate
56  * a document parse.</p>
57  *
58  * <p>All SAX interfaces are assumed to be synchronous: the
59  * {@link #parse parse} methods must not return until parsing
60  * is complete, and readers must wait for an event-handler callback
61  * to return before reporting the next event.</p>
62  *
63  * <p>This interface replaces the (now deprecated) SAX 1.0 {@link
64  * org.xml.sax.Parser Parser} interface. The XMLReader interface
65  * contains two important enhancements over the old Parser
66  * interface (as well as some minor ones):</p>
67  *
68  * <ol>
69  * <li>it adds a standard way to query and set features and
70  * properties; and</li>
71  * <li>it adds Namespace support, which is required for many
72  * higher-level XML standards.</li>
73  * </ol>
74  *
75  * <p>There are adapters available to convert a SAX1 Parser to
76  * a SAX2 XMLReader and vice-versa.</p>
77  *
```

```

78  * @since SAX 2.0
79  * @author David Megginson
80  * @see org.xml.sax.XMLFilter
81  * @see org.xml.sax.helpers.ParserAdapter
82  * @see org.xml.sax.helpers.XMLReaderAdapter
83  */
84  public interface XMLReader
85  {
86
87
88      ///////////////////////////////////////////////////
89      // Configuration.
90      ///////////////////////////////////////////////////
91
92
93      /**
94       * Look up the value of a feature flag.
95       *
96       * <p>The feature name is any fully-qualified URI. It is
97       * possible for an XMLReader to recognize a feature name but
98       * temporarily be unable to return its value.
99       * Some feature values may be available only in specific
100      * contexts, such as before, during, or after a parse.
101      * Also, some feature values may not be programmatically accessible.
102      * (In the case of an adapter for SAX1 {@link Parser}, there is no
103      * implementation-independent way to expose whether the underlying
104      * parser is performing validation, expanding external entities,
105      * and so forth.) </p>
106      *
107      * <p>All XMLReaders are required to recognize the
108      * http://xml.org/sax/features/namespaces and the
109      * http://xml.org/sax/features/namespace-prefixes feature names.</p>
110      *
111      * <p>Typical usage is something like this:</p>
112      *
113      * <pre>
114      * XMLReader r = new MySAXDriver();
115      *
116      *                                     // try to activate validation
117      * try {
118      *     r.setFeature("http://xml.org/sax/features/validation", true);
119      * } catch (SAXException e) {
120      *     System.err.println("Cannot activate validation.");
121      * }
122      *
123      *                                     // register event handlers
124      * r.setContentHandler(new MyContentHandler());
125      * r.setErrorHandler(new MyErrorHandler());
126      *
127      *                                     // parse the first document
128      * try {
129      *     r.parse("http://www.foo.com/mydoc.xml");
130      * } catch (IOException e) {

```

```

131     * System.err.println("I/O exception reading XML document");
132     * } catch (SAXException e) {
133     * System.err.println("XML exception reading document.");
134     * }
135     * </pre>
136     *
137     * <p>Implementors are free (and encouraged) to invent their own features,
138     * using names built on their own URIs.</p>
139     *
140     * @param name The feature name, which is a fully-qualified URI.
141     * @return The current value of the feature (true or false).
142     * @exception org.xml.sax.SAXNotRecognizedException If the feature
143     *         value can't be assigned or retrieved.
144     * @exception org.xml.sax.SAXNotSupportedException When the
145     *         XMLReader recognizes the feature name but
146     *         cannot determine its value at this time.
147     * @see #setFeature
148     */
149     public boolean getFeature (String name)
150         throws SAXNotRecognizedException, SAXNotSupportedException;
151
152
153     /**
154     * Set the value of a feature flag.
155     *
156     * <p>The feature name is any fully-qualified URI. It is
157     * possible for an XMLReader to expose a feature value but
158     * to be unable to change the current value.
159     * Some feature values may be immutable or mutable only
160     * in specific contexts, such as before, during, or after
161     * a parse.</p>
162     *
163     * <p>All XMLReaders are required to support setting
164     * http://xml.org/sax/features/namespaces to true and
165     * http://xml.org/sax/features/namespace-prefixes to false.</p>
166     *
167     * @param name The feature name, which is a fully-qualified URI.
168     * @param value The requested value of the feature (true or false).
169     * @exception org.xml.sax.SAXNotRecognizedException If the feature
170     *         value can't be assigned or retrieved.
171     * @exception org.xml.sax.SAXNotSupportedException When the
172     *         XMLReader recognizes the feature name but
173     *         cannot set the requested value.
174     * @see #getFeature
175     */
176     public void setFeature (String name, boolean value)
177         throws SAXNotRecognizedException, SAXNotSupportedException;
178
179
180     /**
181     * Look up the value of a property.
182     *
183     * <p>The property name is any fully-qualified URI. It is

```

```

184 * possible for an XMLReader to recognize a property name but
185 * temporarily be unable to return its value.
186 * Some property values may be available only in specific
187 * contexts, such as before, during, or after a parse.</p>
188 *
189 * <p>XMLReaders are not required to recognize any specific
190 * property names, though an initial core set is documented for
191 * SAX2.</p>
192 *
193 * <p>Implementors are free (and encouraged) to invent their own properties,
194 * using names built on their own URIs.</p>
195 *
196 * @param name The property name, which is a fully-qualified URI.
197 * @return The current value of the property.
198 * @exception org.xml.sax.SAXNotRecognizedException If the property
199 *         value can't be assigned or retrieved.
200 * @exception org.xml.sax.SAXNotSupportedException When the
201 *         XMLReader recognizes the property name but
202 *         cannot determine its value at this time.
203 * @see #setProperty
204 */
205 public Object getProperty (String name)
206     throws SAXNotRecognizedException, SAXNotSupportedException;
207
208
209 /**
210 * Set the value of a property.
211 *
212 * <p>The property name is any fully-qualified URI. It is
213 * possible for an XMLReader to recognize a property name but
214 * to be unable to change the current value.
215 * Some property values may be immutable or mutable only
216 * in specific contexts, such as before, during, or after
217 * a parse.</p>
218 *
219 * <p>XMLReaders are not required to recognize setting
220 * any specific property names, though a core set is defined by
221 * SAX2.</p>
222 *
223 * <p>This method is also the standard mechanism for setting
224 * extended handlers.</p>
225 *
226 * @param name The property name, which is a fully-qualified URI.
227 * @param value The requested value for the property.
228 * @exception org.xml.sax.SAXNotRecognizedException If the property
229 *         value can't be assigned or retrieved.
230 * @exception org.xml.sax.SAXNotSupportedException When the
231 *         XMLReader recognizes the property name but
232 *         cannot set the requested value.
233 */
234 public void setProperty (String name, Object value)
235     throws SAXNotRecognizedException, SAXNotSupportedException;
236

```

```
237
238
239 ////////////////////////////////////////////////////
240 // Event handlers.
241 ////////////////////////////////////////////////////
242
243
244 /**
245  * Allow an application to register an entity resolver.
246  *
247  * <p>If the application does not register an entity resolver,
248  * the XMLReader will perform its own default resolution.</p>
249  *
250  * <p>Applications may register a new or different resolver in the
251  * middle of a parse, and the SAX parser must begin using the new
252  * resolver immediately.</p>
253  *
254  * @param resolver The entity resolver.
255  * @see #getEntityResolver
256  */
257 public void setEntityResolver (EntityResolver resolver);
258
259
260 /**
261  * Return the current entity resolver.
262  *
263  * @return The current entity resolver, or null if none
264  *         has been registered.
265  * @see #setEntityResolver
266  */
267 public EntityResolver getEntityResolver ();
268
269
270 /**
271  * Allow an application to register a DTD event handler.
272  *
273  * <p>If the application does not register a DTD handler, all DTD
274  * events reported by the SAX parser will be silently ignored.</p>
275  *
276  * <p>Applications may register a new or different handler in the
277  * middle of a parse, and the SAX parser must begin using the new
278  * handler immediately.</p>
279  *
280  * @param handler The DTD handler.
281  * @see #getDTDHandler
282  */
283 public void setDTDHandler (DTDHandler handler);
284
285
286 /**
287  * Return the current DTD handler.
288  *
289  * @return The current DTD handler, or null if none
```



```
290     *      has been registered.
291     * @see #setDTDHandler
292     */
293     public DTDHandler getDTDHandler ();
294
295
296     /**
297     * Allow an application to register a content event handler.
298     *
299     * <p>If the application does not register a content handler, all
300     * content events reported by the SAX parser will be silently
301     * ignored.</p>
302     *
303     * <p>Applications may register a new or different handler in the
304     * middle of a parse, and the SAX parser must begin using the new
305     * handler immediately.</p>
306     *
307     * @param handler The content handler.
308     * @see #getContentHandler
309     */
310     public void setContentHandler (ContentHandler handler);
311
312
313     /**
314     * Return the current content handler.
315     *
316     * @return The current content handler, or null if none
317     *      has been registered.
318     * @see #setContentHandler
319     */
320     public ContentHandler getContentHandler ();
321
322
323     /**
324     * Allow an application to register an error event handler.
325     *
326     * <p>If the application does not register an error handler, all
327     * error events reported by the SAX parser will be silently
328     * ignored; however, normal processing may not continue. It is
329     * highly recommended that all SAX applications implement an
330     * error handler to avoid unexpected bugs.</p>
331     *
332     * <p>Applications may register a new or different handler in the
333     * middle of a parse, and the SAX parser must begin using the new
334     * handler immediately.</p>
335     *
336     * @param handler The error handler.
337     * @see #getErrorHandler
338     */
339     public void setErrorHandler (ErrorHandler handler);
340
341
342     /**
```

```

343     * Return the current error handler.
344     *
345     * @return The current error handler, or null if none
346     *         has been registered.
347     * @see #setErrorHandler
348     */
349     public ErrorHandler getErrorHandler ();
350
351
352
353     //////////////////////////////////////
354     // Parsing.
355     //////////////////////////////////////
356
357     /**
358     * Parse an XML document.
359     *
360     * <p>The application can use this method to instruct the XML
361     * reader to begin parsing an XML document from any valid input
362     * source (a character stream, a byte stream, or a URI).</p>
363     *
364     * <p>Applications may not invoke this method while a parse is in
365     * progress (they should create a new XMLReader instead for each
366     * nested XML document). Once a parse is complete, an
367     * application may reuse the same XMLReader object, possibly with a
368     * different input source.
369     * Configuration of the XMLReader object (such as handler bindings and
370     * values established for feature flags and properties) is unchanged
371     * by completion of a parse, unless the definition of that aspect of
372     * the configuration explicitly specifies other behavior.
373     * (For example, feature flags or properties exposing
374     * characteristics of the document being parsed.)
375     * </p>
376     *
377     * <p>During the parse, the XMLReader will provide information
378     * about the XML document through the registered event
379     * handlers.</p>
380     *
381     * <p>This method is synchronous: it will not return until parsing
382     * has ended. If a client application wants to terminate
383     * parsing early, it should throw an exception.</p>
384     *
385     * @param input The input source for the top-level of the
386     *             XML document.
387     * @exception org.xml.sax.SAXException Any SAX exception, possibly
388     *             wrapping another exception.
389     * @exception java.io.IOException An IO exception from the parser,
390     *             possibly from a byte stream or character stream
391     *             supplied by the application.
392     * @see org.xml.sax.InputSource
393     * @see #parse(java.lang.String)
394     * @see #setEntityResolver
395     * @see #setDTDHandler

```

```

396     * @see #setContentHandler
397     * @see #setErrorHandler
398     */
399     public void parse (InputSource input)
400         throws IOException, SAXException;
401
402
403     /**
404     * Parse an XML document from a system identifier (URI).
405     *
406     * <p>This method is a shortcut for the common case of reading a
407     * document from a system identifier. It is the exact
408     * equivalent of the following:</p>
409     *
410     * <pre>
411     * parse(new InputSource(systemId));
412     * </pre>
413     *
414     * <p>If the system identifier is a URL, it must be fully resolved
415     * by the application before it is passed to the parser.</p>
416     *
417     * @param systemId The system identifier (URI).
418     * @exception org.xml.sax.SAXException Any SAX exception, possibly
419     *         wrapping another exception.
420     * @exception java.io.IOException An IO exception from the parser,
421     *         possibly from a byte stream or character stream
422     *         supplied by the application.
423     * @see #parse(org.xml.sax.InputSource)
424     */
425     public void parse (String systemId)
426         throws IOException, SAXException;
427
428 }

```

## 48 Sax Ext (org.xml.sax.ext package)

### 48.1 Attributes2.java

Secuencia 495: org/xml/sax/ext/Attributes2.java

```

1  /*
2  * Copyright (c) 2004, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```

```

15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  // Attributes2.java - extended Attributes
27  // http://www.saxproject.org
28  // Public Domain: no warranty.
29  // $Id: Attributes2.java,v 1.2 2004/11/03 22:49:07 jsuttor Exp $
30
31  package org.xml.sax.ext;
32
33  import org.xml.sax.Attributes;
34
35
36  /**
37   * SAX2 extension to augment the per-attribute information
38   * provided though {@link Attributes}.
39   * If an implementation supports this extension, the attributes
40   * provided in {@link org.xml.sax.ContentHandler#startElement
41   * ContentHandler.startElement() } will implement this interface,
42   * and the <em>http://xml.org/sax/features/use-attributes2</em>
43   * feature flag will have the value <em>true</em>.
44   *
45   * <blockquote>
46   * <em>This module, both source code and documentation, is in the
47   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
48   * </blockquote>
49   *
50   * <p>XMLReader implementations are not required to support this
51   * information, and it is not part of core-only SAX2 distributions.</p>
52   *
53   * <p>Note that if an attribute was defaulted (<em>!isSpecified()</em>)
54   * it will of necessity also have been declared (<em>isDeclared()</em>)
55   * in the DTD.
56   * Similarly if an attribute's type is anything except CDATA, then it
57   * must have been declared.
58   * </p>
59   *
60   * @since SAX 2.0 (extensions 1.1 alpha)
61   * @author David Brownell
62   */
63  public interface Attributes2 extends Attributes
64  {
65      /**
66       * Returns false unless the attribute was declared in the DTD.
67       * This helps distinguish two kinds of attributes that SAX reports

```

```
68      * as CDATA: ones that were declared (and hence are usually valid),
69      * and those that were not (and which are never valid).
70      *
71      * @param index The attribute index (zero-based).
72      * @return true if the attribute was declared in the DTD,
73      *         false otherwise.
74      * @exception java.lang.ArrayIndexOutOfBoundsException When the
75      *         supplied index does not identify an attribute.
76      */
77     public boolean isDeclared (int index);
78
79     /**
80      * Returns false unless the attribute was declared in the DTD.
81      * This helps distinguish two kinds of attributes that SAX reports
82      * as CDATA: ones that were declared (and hence are usually valid),
83      * and those that were not (and which are never valid).
84      *
85      * @param qName The XML qualified (prefixed) name.
86      * @return true if the attribute was declared in the DTD,
87      *         false otherwise.
88      * @exception java.lang.IllegalArgumentException When the
89      *         supplied name does not identify an attribute.
90      */
91     public boolean isDeclared (String qName);
92
93     /**
94      * Returns false unless the attribute was declared in the DTD.
95      * This helps distinguish two kinds of attributes that SAX reports
96      * as CDATA: ones that were declared (and hence are usually valid),
97      * and those that were not (and which are never valid).
98      *
99      * <p>Remember that since DTDs do not "understand" namespaces, the
100     * namespace URI associated with an attribute may not have come from
101     * the DTD. The declaration will have applied to the attribute's
102     * <em>qName</em>.
103     *
104     * @param uri The Namespace URI, or the empty string if
105     *             the name has no Namespace URI.
106     * @param localName The attribute's local name.
107     * @return true if the attribute was declared in the DTD,
108     *         false otherwise.
109     * @exception java.lang.IllegalArgumentException When the
110     *         supplied names do not identify an attribute.
111     */
112     public boolean isDeclared (String uri, String localName);
113
114     /**
115      * Returns true unless the attribute value was provided
116      * by DTD defaulting.
117      *
118      * @param index The attribute index (zero-based).
119      * @return true if the value was found in the XML text,
120      *         false if the value was provided by DTD defaulting.
```

```

121     * @exception java.lang.ArrayIndexOutOfBoundsException When the
122     *         supplied index does not identify an attribute.
123     */
124     public boolean isSpecified (int index);
125
126     /**
127     * Returns true unless the attribute value was provided
128     * by DTD defaulting.
129     *
130     * <p>Remember that since DTDs do not "understand" namespaces, the
131     * namespace URI associated with an attribute may not have come from
132     * the DTD. The declaration will have applied to the attribute's
133     * <em>qName</em>.
134     *
135     * @param uri The Namespace URI, or the empty string if
136     *         the name has no Namespace URI.
137     * @param localName The attribute's local name.
138     * @return true if the value was found in the XML text,
139     *         false if the value was provided by DTD defaulting.
140     * @exception java.lang.IllegalArgumentException When the
141     *         supplied names do not identify an attribute.
142     */
143     public boolean isSpecified (String uri, String localName);
144
145     /**
146     * Returns true unless the attribute value was provided
147     * by DTD defaulting.
148     *
149     * @param qName The XML qualified (prefixed) name.
150     * @return true if the value was found in the XML text,
151     *         false if the value was provided by DTD defaulting.
152     * @exception java.lang.IllegalArgumentException When the
153     *         supplied name does not identify an attribute.
154     */
155     public boolean isSpecified (String qName);
156 }

```

## 48.2 Attributes2Impl.java

Secuencia 496: org/xml/sax/ext/Attributes2Impl.java

```

1  /*
2  * Copyright (c) 2004, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *

```

```
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  // Attributes2Impl.java - extended AttributesImpl
27  // http://www.saxproject.org
28  // Public Domain: no warranty.
29  // $Id: Attributes2Impl.java,v 1.3 2005/02/24 11:20:18 gg156739 Exp $
30
31  package org.xml.sax.ext;
32
33  import org.xml.sax.Attributes;
34  import org.xml.sax.helpers.AttributesImpl;
35
36
37  /**
38   * SAX2 extension helper for additional Attributes information,
39   * implementing the {@link Attributes2} interface.
40   *
41   * <blockquote>
42   * <em>This module, both source code and documentation, is in the
43   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
44   * </blockquote>
45   *
46   * <p>This is not part of core-only SAX2 distributions.</p>
47   *
48   * <p>The <em>specified</em> flag for each attribute will always
49   * be true, unless it has been set to false in the copy constructor
50   * or using {@link #setSpecified}.
51   * Similarly, the <em>declared</em> flag for each attribute will
52   * always be false, except for defaulted attributes (<em>specified</em>
53   * is false), non-CDATA attributes, or when it is set to true using
54   * {@link #setDeclared}.
55   * If you change an attribute's type by hand, you may need to modify
56   * its <em>declared</em> flag to match.
57   * </p>
58   *
59   * @since SAX 2.0 (extensions 1.1 alpha)
60   * @author David Brownell
61   */
62  public class Attributes2Impl extends AttributesImpl implements Attributes2
63  {
64      private boolean    declared [];
65      private boolean    specified [];
66  }
```

```

67
68 /**
69  * Construct a new, empty Attributes2Impl object.
70  */
71 public Attributes2Impl () {
72     specified = null;
73     declared = null;
74 }
75
76
77 /**
78  * Copy an existing Attributes or Attributes2 object.
79  * If the object implements Attributes2, values of the
80  * <em>specified</em> and <em>declared</em> flags for each
81  * attribute are copied.
82  * Otherwise the flag values are defaulted to assume no DTD was used,
83  * unless there is evidence to the contrary (such as attributes with
84  * type other than CDATA, which must have been <em>declared</em>).
85  *
86  * <p>This constructor is especially useful inside a
87  * {@link org.xml.sax.ContentHandler#startElement startElement} event.</p>
88  *
89  * @param atts The existing Attributes object.
90  */
91 public Attributes2Impl (Attributes atts)
92 {
93     super (atts);
94 }
95
96
97 ////////////////////////////////////////////////////
98 // Implementation of Attributes2
99 ////////////////////////////////////////////////////
100
101
102 /**
103  * Returns the current value of the attribute's "declared" flag.
104  */
105 // javadoc mostly from interface
106 public boolean isDeclared (int index)
107 {
108     if (index < 0 || index >= getLength ())
109         throw new ArrayIndexOutOfBoundsException (
110             "No attribute at index: " + index);
111     return declared [index];
112 }
113
114
115 /**
116  * Returns the current value of the attribute's "declared" flag.
117  */
118 // javadoc mostly from interface
119 public boolean isDeclared (String uri, String localName)

```



```
120 {
121     int index = getIndex (uri, localName);
122
123     if (index < 0)
124         throw new IllegalArgumentException (
125             "No such attribute: local=" + localName
126             + ", namespace=" + uri);
127     return declared [index];
128 }
129
130
131 /**
132  * Returns the current value of the attribute's "declared" flag.
133  */
134 // javadoc mostly from interface
135 public boolean isDeclared (String qName)
136 {
137     int index = getIndex (qName);
138
139     if (index < 0)
140         throw new IllegalArgumentException (
141             "No such attribute: " + qName);
142     return declared [index];
143 }
144
145
146 /**
147  * Returns the current value of an attribute's "specified" flag.
148  *
149  * @param index The attribute index (zero-based).
150  * @return current flag value
151  * @exception java.lang.ArrayIndexOutOfBoundsException When the
152  *         supplied index does not identify an attribute.
153  */
154 public boolean isSpecified (int index)
155 {
156     if (index < 0 || index >= getLength ())
157         throw new ArrayIndexOutOfBoundsException (
158             "No attribute at index: " + index);
159     return specified [index];
160 }
161
162
163 /**
164  * Returns the current value of an attribute's "specified" flag.
165  *
166  * @param uri The Namespace URI, or the empty string if
167  *         the name has no Namespace URI.
168  * @param localName The attribute's local name.
169  * @return current flag value
170  * @exception java.lang.IllegalArgumentException When the
171  *         supplied names do not identify an attribute.
172  */
```

```

173 public boolean isSpecified (String uri, String localName)
174 {
175     int index = getIndex (uri, localName);
176
177     if (index < 0)
178         throw new IllegalArgumentException (
179             "No such attribute: local=" + localName
180             + ", namespace=" + uri);
181     return specified [index];
182 }
183
184
185 /**
186  * Returns the current value of an attribute's "specified" flag.
187  *
188  * @param qName The XML qualified (prefixed) name.
189  * @return current flag value
190  * @exception java.lang.IllegalArgumentException When the
191  *         supplied name does not identify an attribute.
192  */
193 public boolean isSpecified (String qName)
194 {
195     int index = getIndex (qName);
196
197     if (index < 0)
198         throw new IllegalArgumentException (
199             "No such attribute: " + qName);
200     return specified [index];
201 }
202
203
204 ////////////////////////////////////////////////////
205 // Manipulators
206 ////////////////////////////////////////////////////
207
208
209 /**
210  * Copy an entire Attributes object. The "specified" flags are
211  * assigned as true, and "declared" flags as false (except when
212  * an attribute's type is not CDATA),
213  * unless the object is an Attributes2 object.
214  * In that case those flag values are all copied.
215  *
216  * @see AttributesImpl#setAttributes
217  */
218 public void setAttributes (Attributes atts)
219 {
220     int length = atts.getLength ();
221
222     super.setAttributes (atts);
223     declared = new boolean [length];
224     specified = new boolean [length];
225

```

```

226     if (atts instanceof Attributes2) {
227         Attributes2 a2 = (Attributes2) atts;
228         for (int i = 0; i < length; i++) {
229             declared [i] = a2.isDeclared (i);
230             specified [i] = a2.isSpecified (i);
231         }
232     } else {
233         for (int i = 0; i < length; i++) {
234             declared [i] = !"CDATA".equals (atts.getType (i));
235             specified [i] = true;
236         }
237     }
238 }
239
240
241 /**
242  * Add an attribute to the end of the list, setting its
243  * "specified" flag to true. To set that flag's value
244  * to false, use {@link #setSpecified}.
245  *
246  * <p>Unless the attribute <em>type</em> is CDATA, this attribute
247  * is marked as being declared in the DTD. To set that flag's value
248  * to true for CDATA attributes, use {@link #setDeclared}.
249  *
250  * @see AttributesImpl#addAttribute
251  */
252 public void addAttribute (String uri, String localName, String qName,
253                          String type, String value)
254 {
255     super.addAttribute (uri, localName, qName, type, value);
256
257
258     int length = getLength ();
259     if(specified==null)
260     {
261         specified = new boolean[length];
262         declared = new boolean[length];
263     } else if (length > specified.length) {
264         boolean    newFlags [];
265
266         newFlags = new boolean [length];
267         System.arraycopy (declared, 0, newFlags, 0, declared.length);
268         declared = newFlags;
269
270         newFlags = new boolean [length];
271         System.arraycopy (specified, 0, newFlags, 0, specified.length);
272         specified = newFlags;
273     }
274
275     specified [length - 1] = true;
276     declared [length - 1] = !"CDATA".equals (type);
277 }
278

```

```

279
280 // javadoc entirely from superclass
281 public void removeAttribute (int index)
282 {
283     int origMax = getLength () - 1;
284
285     super.removeAttribute (index);
286     if (index != origMax) {
287         System.arraycopy (declared, index + 1, declared, index,
288             origMax - index);
289         System.arraycopy (specified, index + 1, specified, index,
290             origMax - index);
291     }
292 }
293
294
295 /**
296  * Assign a value to the "declared" flag of a specific attribute.
297  * This is normally needed only for attributes of type CDATA,
298  * including attributes whose type is changed to or from CDATA.
299  *
300  * @param index The index of the attribute (zero-based).
301  * @param value The desired flag value.
302  * @exception java.lang.ArrayIndexOutOfBoundsException When the
303  *             supplied index does not identify an attribute.
304  * @see #setType
305  */
306 public void setDeclared (int index, boolean value)
307 {
308     if (index < 0 || index >= getLength ())
309         throw new ArrayIndexOutOfBoundsException (
310             "No attribute at index: " + index);
311     declared [index] = value;
312 }
313
314
315 /**
316  * Assign a value to the "specified" flag of a specific attribute.
317  * This is the only way this flag can be cleared, except clearing
318  * by initialization with the copy constructor.
319  *
320  * @param index The index of the attribute (zero-based).
321  * @param value The desired flag value.
322  * @exception java.lang.ArrayIndexOutOfBoundsException When the
323  *             supplied index does not identify an attribute.
324  */
325 public void setSpecified (int index, boolean value)
326 {
327     if (index < 0 || index >= getLength ())
328         throw new ArrayIndexOutOfBoundsException (
329             "No attribute at index: " + index);
330     specified [index] = value;
331 }

```

332 }

### 48.3 DeclHandler.java

Secuencia 497: org/xml/sax/ext/DeclHandler.java

```
1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // DeclHandler.java - Optional handler for DTD declaration events.
27 // http://www.saxproject.org
28 // Public Domain: no warranty.
29 // $Id: DeclHandler.java,v 1.2 2004/11/03 22:49:08 jsuttor Exp $
30
31 package org.xml.sax.ext;
32
33 import org.xml.sax.SAXException;
34
35
36 /**
37 * SAX2 extension handler for DTD declaration events.
38 *
39 * <blockquote>
40 * <em>This module, both source code and documentation, is in the
41 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
42 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
43 * for further information.
44 * </blockquote>
45 *
46 * <p>This is an optional extension handler for SAX2 to provide more
47 * complete information about DTD declarations in an XML document.
48 * XML readers are not required to recognize this handler, and it
```

```

49  * is not part of core-only SAX2 distributions.</p>
50  *
51  * <p>Note that data-related DTD declarations (unparsed entities and
52  * notations) are already reported through the {@link
53  * org.xml.sax.DTDHandler DTDHandler} interface.</p>
54  *
55  * <p>If you are using the declaration handler together with a lexical
56  * handler, all of the events will occur between the
57  * {@link org.xml.sax.ext.LexicalHandler#startDTD startDTD} and the
58  * {@link org.xml.sax.ext.LexicalHandler#endDTD endDTD} events.</p>
59  *
60  * <p>To set the DeclHandler for an XML reader, use the
61  * {@link org.xml.sax.XMLReader#setProperty setProperty} method
62  * with the property name
63  * <code>http://xml.org/sax/properties/declaration-handler</code>
64  * and an object implementing this interface (or null) as the value.
65  * If the reader does not report declaration events, it will throw a
66  * {@link org.xml.sax.SAXNotRecognizedException SAXNotRecognizedException}
67  * when you attempt to register the handler.</p>
68  *
69  * @since SAX 2.0 (extensions 1.0)
70  * @author David Megginson
71  */
72  public interface DeclHandler
73  {
74
75      /**
76       * Report an element type declaration.
77       *
78       * <p>The content model will consist of the string "EMPTY", the
79       * string "ANY", or a parenthesised group, optionally followed
80       * by an occurrence indicator. The model will be normalized so
81       * that all parameter entities are fully resolved and all whitespace
82       * is removed, and will include the enclosing parentheses. Other
83       * normalization (such as removing redundant parentheses or
84       * simplifying occurrence indicators) is at the discretion of the
85       * parser.</p>
86       *
87       * @param name The element type name.
88       * @param model The content model as a normalized string.
89       * @exception SAXException The application may raise an exception.
90       */
91      public abstract void elementDecl (String name, String model)
92          throws SAXException;
93
94
95      /**
96       * Report an attribute type declaration.
97       *
98       * <p>Only the effective (first) declaration for an attribute will
99       * be reported. The type will be one of the strings "CDATA",
100      * "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY",
101      * "ENTITIES", a parenthesized token group with

```

```

102 * the separator "|" and all whitespace removed, or the word
103 * "NOTATION" followed by a space followed by a parenthesized
104 * token group with all whitespace removed.</p>
105 *
106 * <p>The value will be the value as reported to applications,
107 * appropriately normalized and with entity and character
108 * references expanded. </p>
109 *
110 * @param eName The name of the associated element.
111 * @param aName The name of the attribute.
112 * @param type A string representing the attribute type.
113 * @param mode A string representing the attribute defaulting mode
114 *      ("IMPLIED", "REQUIRED", or "FIXED") or null if
115 *      none of these applies.
116 * @param value A string representing the attribute's default value,
117 *      or null if there is none.
118 * @exception SAXException The application may raise an exception.
119 */
120 public abstract void attributeDecl (String eName,
121                                     String aName,
122                                     String type,
123                                     String mode,
124                                     String value)
125     throws SAXException;
126
127
128 /**
129  * Report an internal entity declaration.
130  *
131  * <p>Only the effective (first) declaration for each entity
132  * will be reported. All parameter entities in the value
133  * will be expanded, but general entities will not.</p>
134  *
135  * @param name The name of the entity. If it is a parameter
136  *      entity, the name will begin with '%'.
137  * @param value The replacement text of the entity.
138  * @exception SAXException The application may raise an exception.
139  * @see #externalEntityDecl
140  * @see org.xml.sax.DTDHandler#unparsedEntityDecl
141  */
142 public abstract void internalEntityDecl (String name, String value)
143     throws SAXException;
144
145
146 /**
147  * Report a parsed external entity declaration.
148  *
149  * <p>Only the effective (first) declaration for each entity
150  * will be reported.</p>
151  *
152  * <p>If the system identifier is a URL, the parser must resolve it
153  * fully before passing it to the application.</p>
154  *

```

```

155     * @param name The name of the entity. If it is a parameter
156     *           entity, the name will begin with '%'.
157     * @param publicId The entity's public identifier, or null if none
158     *           was given.
159     * @param systemId The entity's system identifier.
160     * @exception SAXException The application may raise an exception.
161     * @see #internalEntityDecl
162     * @see org.xml.sax.DTDHandler#unparsedEntityDecl
163     */
164     public abstract void externalEntityDecl (String name, String publicId,
165                                             String systemId)
166         throws SAXException;
167
168 }
169
170 // end of DeclHandler.java

```

#### 48.4 DefaultHandler2.java

Secuencia 498: org/xml/sax/ext/DefaultHandler2.java

```

1  /*
2  * Copyright (c) 2004, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // DefaultHandler2.java - extended DefaultHandler
27 // http://www.saxproject.org
28 // Public Domain: no warranty.
29 // $Id: DefaultHandler2.java,v 1.2 2004/11/03 22:49:08 jsuttor Exp $
30
31 package org.xml.sax.ext;
32
33 import java.io.IOException;

```



```

34 import org.xml.sax.InputSource;
35 import org.xml.sax.SAXException;
36 import org.xml.sax.helpers.DefaultHandler;
37
38
39 /**
40  * This class extends the SAX2 base handler class to support the
41  * SAX2 {@link LexicalHandler}, {@link DeclHandler}, and
42  * {@link EntityResolver2} extensions. Except for overriding the
43  * original SAX1 {@link DefaultHandler#resolveEntity resolveEntity()}
44  * method the added handler methods just return. Subclassers may
45  * override everything on a method-by-method basis.
46  *
47  * <blockquote>
48  * <em>This module, both source code and documentation, is in the
49  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
50  * </blockquote>
51  *
52  * <p> <em>Note:</em> this class might yet learn that the
53  * <em>ContentHandler.setDocumentLocator()</em> call might be passed a
54  * {@link Locator2} object, and that the
55  * <em>ContentHandler.startElement()</em> call might be passed a
56  * {@link Attributes2} object.
57  *
58  * @since SAX 2.0 (extensions 1.1 alpha)
59  * @author David Brownell
60  */
61 public class DefaultHandler2 extends DefaultHandler
62     implements LexicalHandler, DeclHandler, EntityResolver2
63 {
64     /** Constructs a handler which ignores all parsing events. */
65     public DefaultHandler2 () { }
66
67
68     // SAX2 ext-1.0 LexicalHandler
69
70     public void startCDATA ()
71     throws SAXException
72     {}
73
74     public void endCDATA ()
75     throws SAXException
76     {}
77
78     public void startDTD (String name, String publicId, String systemId)
79     throws SAXException
80     {}
81
82     public void endDTD ()
83     throws SAXException
84     {}
85
86     public void startEntity (String name)

```

[illegible]

```

140     throws SAXException, IOException
141     { return null; }
142
143     // SAX1 EntityResolver
144
145     /**
146     * Invokes
147     * {@link EntityResolver2#resolveEntity EntityResolver2.resolveEntity()}
148     * with null entity name and base URI.
149     * You only need to override that method to use this class.
150     */
151     public InputSource resolveEntity (String publicId, String systemId)
152     throws SAXException, IOException
153     { return resolveEntity (null, publicId, null, systemId); }
154 }

```

## 48.5 EntityResolver2.java

Secuencia 499: org/xml/sax/ext/EntityResolver2.java

```

1  /*
2  * Copyright (c) 2004, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // EntityResolver2.java - Extended SAX entity resolver.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: EntityResolver2.java,v 1.2 2004/11/03 22:49:08 jsuttor Exp $
30
31 package org.xml.sax.ext;
32
33 import java.io.IOException;
34

```

```

35 import org.xml.sax.EntityResolver;
36 import org.xml.sax.InputSource;
37 import org.xml.sax.XMLReader;
38 import org.xml.sax.SAXException;
39
40
41 /**
42  * Extended interface for mapping external entity references to input
43  * sources, or providing a missing external subset. The
44  * {@link XMLReader#setEntityResolver XMLReader.setEntityResolver()} method
45  * is used to provide implementations of this interface to parsers.
46  * When a parser uses the methods in this interface, the
47  * {@link EntityResolver2#resolveEntity EntityResolver2.resolveEntity()}
48  * method (in this interface) is used instead of the older (SAX 1.0)
49  * {@link EntityResolver#resolveEntity EntityResolver.resolveEntity()} method.
50  *
51  * <blockquote>
52  * <em>This module, both source code and documentation, is in the
53  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
54  * </blockquote>
55  *
56  * <p>If a SAX application requires the customized handling which this
57  * interface defines for external entities, it must ensure that it uses
58  * an XMLReader with the
59  * <em>http://xml.org/sax/features/use-entity-resolver2</em> feature flag
60  * set to <em>true</em> (which is its default value when the feature is
61  * recognized). If that flag is unrecognized, or its value is false,
62  * or the resolver does not implement this interface, then only the
63  * {@link EntityResolver} method will be used.
64  * </p>
65  *
66  * <p>That supports three categories of application that modify entity
67  * resolution. <em>Old Style</em> applications won't know about this interface;
68  * they will provide an EntityResolver.
69  * <em>Transitional Mode</em> provide an EntityResolver2 and automatically
70  * get the benefit of its methods in any systems (parsers or other tools)
71  * supporting it, due to polymorphism.
72  * Both <em>Old Style</em> and <em>Transitional Mode</em> applications will
73  * work with any SAX2 parser.
74  * <em>New style</em> applications will fail to run except on SAX2 parsers
75  * that support this particular feature.
76  * They will insist that feature flag have a value of "true", and the
77  * EntityResolver2 implementation they provide might throw an exception
78  * if the original SAX 1.0 style entity resolution method is invoked.
79  * </p>
80  *
81  * @see org.xml.sax.XMLReader#setEntityResolver
82  *
83  * @since SAX 2.0 (extensions 1.1 alpha)
84  * @author David Brownell
85  */
86 public interface EntityResolver2 extends EntityResolver
87 {

```

```

88  /**
89  * Allows applications to provide an external subset for documents
90  * that don't explicitly define one. Documents with DOCTYPE declarations
91  * that omit an external subset can thus augment the declarations
92  * available for validation, entity processing, and attribute processing
93  * (normalization, defaulting, and reporting types including ID).
94  * This augmentation is reported
95  * through the {@link LexicalHandler#startDTD startDTD()} method as if
96  * the document text had originally included the external subset;
97  * this callback is made before any internal subset data or errors
98  * are reported.</p>
99  *
100 * <p>This method can also be used with documents that have no DOCTYPE
101 * declaration. When the root element is encountered,
102 * but no DOCTYPE declaration has been seen, this method is
103 * invoked. If it returns a value for the external subset, that root
104 * element is declared to be the root element, giving the effect of
105 * splicing a DOCTYPE declaration at the end the prolog of a document
106 * that could not otherwise be valid. The sequence of parser callbacks
107 * in that case logically resembles this:</p>
108 *
109 * <pre>
110 * ... comments and PIs from the prolog (as usual)
111 * startDTD ("rootName", source.getPublicId (), source.getSystemId ());
112 * startEntity ("[dtd]");
113 * ... declarations, comments, and PIs from the external subset
114 * endEntity ("[dtd]");
115 * endDTD ();
116 * ... then the rest of the document (as usual)
117 * startElement (... , "rootName", ...);
118 * </pre>
119 *
120 * <p>Note that the InputSource gets no further resolution.
121 * Implementations of this method may wish to invoke
122 * {@link #resolveEntity resolveEntity()} to gain benefits such as use
123 * of local caches of DTD entities. Also, this method will never be
124 * used by a (non-validating) processor that is not including external
125 * parameter entities. </p>
126 *
127 * <p>Uses for this method include facilitating data validation when
128 * interoperating with XML processors that would always require
129 * undesirable network accesses for external entities, or which for
130 * other reasons adopt a "no DTDs" policy.
131 * Non-validation motives include forcing documents to include DTDs so
132 * that attributes are handled consistently.
133 * For example, an XPath processor needs to know which attributes have
134 * type "ID" before it can process a widely used type of reference.</p>
135 *
136 * <p><strong>Warning:</strong> Returning an external subset modifies
137 * the input document. By providing definitions for general entities,
138 * it can make a malformed document appear to be well formed.
139 * </p>
140 *

```

```

141 * @param name Identifies the document root element. This name comes
142 * from a DOCTYPE declaration (where available) or from the actual
143 * root element.
144 * @param baseURI The document's base URI, serving as an additional
145 * hint for selecting the external subset. This is always an absolute
146 * URI, unless it is null because the XMLReader was given an InputSource
147 * without one.
148 *
149 * @return An InputSource object describing the new external subset
150 * to be used by the parser, or null to indicate that no external
151 * subset is provided.
152 *
153 * @exception SAXException Any SAX exception, possibly wrapping
154 * another exception.
155 * @exception IOException Probably indicating a failure to create
156 * a new InputStream or Reader, or an illegal URL.
157 */
158 public InputSource getExternalSubset (String name, String baseURI)
159 throws SAXException, IOException;
160
161 /**
162 * Allows applications to map references to external entities into input
163 * sources, or tell the parser it should use conventional URI resolution.
164 * This method is only called for external entities which have been
165 * properly declared.
166 * This method provides more flexibility than the {@link EntityResolver}
167 * interface, supporting implementations of more complex catalogue
168 * schemes such as the one defined by the <a href=
169 * "http://www.oasis-open.org/committees/entity/spec-2001-08-06.html"
170 * >OASIS XML Catalogs</a> specification.</p>
171 *
172 * <p>Parsers configured to use this resolver method will call it
173 * to determine the input source to use for any external entity
174 * being included because of a reference in the XML text.
175 * That excludes the document entity, and any external entity returned
176 * by {@link #getExternalSubset getExternalSubset()}.
177 * When a (non-validating) processor is configured not to include
178 * a class of entities (parameter or general) through use of feature
179 * flags, this method is not invoked for such entities. </p>
180 *
181 * <p>Note that the entity naming scheme used here is the same one
182 * used in the {@link LexicalHandler}, or in the {@link
183 * org.xml.sax.ContentHandler#skippedEntity
184 * ContentHandler.skippedEntity()}
185 * method. </p>
186 *
187 * @param name Identifies the external entity being resolved.
188 * Either "[dtd]" for the external subset, or a name starting
189 * with "%" to indicate a parameter entity, or else the name of
190 * a general entity. This is never null when invoked by a SAX2
191 * parser.
192 * @param publicId The public identifier of the external entity being
193 * referenced (normalized as required by the XML specification), or

```

```

194     * null if none was supplied.
195     * @param baseURI The URI with respect to which relative systemIDs
196     * are interpreted. This is always an absolute URI, unless it is
197     * null (likely because the XMLReader was given an InputSource without
198     * one). This URI is defined by the XML specification to be the one
199     * associated with the "<" starting the relevant declaration.
200     * @param systemId The system identifier of the external entity
201     * being referenced; either a relative or absolute URI.
202     * This is never null when invoked by a SAX2 parser; only declared
203     * entities, and any external subset, are resolved by such parsers.
204     *
205     * @return An InputSource object describing the new input source to
206     * be used by the parser. Returning null directs the parser to
207     * resolve the system ID against the base URI and open a connection
208     * to resulting URI.
209     *
210     * @exception SAXException Any SAX exception, possibly wrapping
211     * another exception.
212     * @exception IOException Probably indicating a failure to create
213     * a new InputStream or Reader, or an illegal URL.
214     */
215     public InputSource resolveEntity (
216         String name,
217         String publicId,
218         String baseURI,
219         String systemId
220     ) throws SAXException, IOException;
221 }

```

## 48.6 LexicalHandler.java

Secuencia 500: org/xml/sax/ext/LexicalHandler.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *

```

```
22  *
23  *
24  */
25
26  // LexicalHandler.java - optional handler for lexical parse events.
27  // http://www.saxproject.org
28  // Public Domain: no warranty.
29  // $Id: LexicalHandler.java,v 1.2 2004/11/03 22:49:08 jsuttor Exp $
30
31  package org.xml.sax.ext;
32
33  import org.xml.sax.SAXException;
34
35  /**
36   * SAX2 extension handler for lexical events.
37   *
38   * <blockquote>
39   * <em>This module, both source code and documentation, is in the
40   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
41   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
42   * for further information.
43   * </blockquote>
44   *
45   * <p>This is an optional extension handler for SAX2 to provide
46   * lexical information about an XML document, such as comments
47   * and CDATA section boundaries.
48   * XML readers are not required to recognize this handler, and it
49   * is not part of core-only SAX2 distributions.</p>
50   *
51   * <p>The events in the lexical handler apply to the entire document,
52   * not just to the document element, and all lexical handler events
53   * must appear between the content handler's startDocument and
54   * endDocument events.</p>
55   *
56   * <p>To set the LexicalHandler for an XML reader, use the
57   * {@link org.xml.sax.XMLReader#setProperty setProperty} method
58   * with the property name
59   * <code>http://xml.org/sax/properties/lexical-handler</code>
60   * and an object implementing this interface (or null) as the value.
61   * If the reader does not report lexical events, it will throw a
62   * {@link org.xml.sax.SAXNotRecognizedException SAXNotRecognizedException}
63   * when you attempt to register the handler.</p>
64   *
65   * @since SAX 2.0 (extensions 1.0)
66   * @author David Megginson
67   */
68  public interface LexicalHandler
69  {
70
71      /**
72       * Report the start of DTD declarations, if any.
73       *
74       * <p>This method is intended to report the beginning of the
```



```

75      * DOCTYPE declaration; if the document has no DOCTYPE declaration,
76      * this method will not be invoked.</p>
77      *
78      * <p>All declarations reported through
79      * {@link org.xml.sax.DTDHandler DTDHandler} or
80      * {@link org.xml.sax.ext.DeclHandler DeclHandler} events must appear
81      * between the startDTD and {@link #endDTD endDTD} events.
82      * Declarations are assumed to belong to the internal DTD subset
83      * unless they appear between {@link #startEntity startEntity}
84      * and {@link #endEntity endEntity} events. Comments and
85      * processing instructions from the DTD should also be reported
86      * between the startDTD and endDTD events, in their original
87      * order of (logical) occurrence; they are not required to
88      * appear in their correct locations relative to DTDHandler
89      * or DeclHandler events, however.</p>
90      *
91      * <p>Note that the start/endDTD events will appear within
92      * the start/endDocument events from ContentHandler and
93      * before the first
94      * {@link org.xml.sax.ContentHandler#startElement startElement}
95      * event.</p>
96      *
97      * @param name The document type name.
98      * @param publicId The declared public identifier for the
99      *     external DTD subset, or null if none was declared.
100     * @param systemId The declared system identifier for the
101     *     external DTD subset, or null if none was declared.
102     *     (Note that this is not resolved against the document
103     *     base URI.)
104     * @exception SAXException The application may raise an
105     *     exception.
106     * @see #endDTD
107     * @see #startEntity
108     */
109     public abstract void startDTD (String name, String publicId,
110                                   String systemId)
111         throws SAXException;
112
113
114     /**
115     * Report the end of DTD declarations.
116     *
117     * <p>This method is intended to report the end of the
118     * DOCTYPE declaration; if the document has no DOCTYPE declaration,
119     * this method will not be invoked.</p>
120     *
121     * @exception SAXException The application may raise an exception.
122     * @see #startDTD
123     */
124     public abstract void endDTD ()
125         throws SAXException;
126
127

```

```

128 /**
129  * Report the beginning of some internal and external XML entities.
130  *
131  * <p>The reporting of parameter entities (including
132  * the external DTD subset) is optional, and SAX2 drivers that
133  * report LexicalHandler events may not implement it; you can use the
134  * <code>
135  * >http://xml.org/sax/features/lexical-handler/parameter-entities</code>
136  * feature to query or control the reporting of parameter entities.</p>
137  *
138  * <p>General entities are reported with their regular names,
139  * parameter entities have '%' prepended to their names, and
140  * the external DTD subset has the pseudo-entity name "[dtd]".</p>
141  *
142  * <p>When a SAX2 driver is providing these events, all other
143  * events must be properly nested within start/end entity
144  * events. There is no additional requirement that events from
145  * {@link org.xml.sax.ext.DeclHandler DeclHandler} or
146  * {@link org.xml.sax.DTDHandler DTDHandler} be properly ordered.</p>
147  *
148  * <p>Note that skipped entities will be reported through the
149  * {@link org.xml.sax.ContentHandler#skippedEntity skippedEntity}
150  * event, which is part of the ContentHandler interface.</p>
151  *
152  * <p>Because of the streaming event model that SAX uses, some
153  * entity boundaries cannot be reported under any
154  * circumstances:</p>
155  *
156  * <ul>
157  * <li>general entities within attribute values</li>
158  * <li>parameter entities within declarations</li>
159  * </ul>
160  *
161  * <p>These will be silently expanded, with no indication of where
162  * the original entity boundaries were.</p>
163  *
164  * <p>Note also that the boundaries of character references (which
165  * are not really entities anyway) are not reported.</p>
166  *
167  * <p>All start/endEntity events must be properly nested.
168  *
169  * @param name The name of the entity. If it is a parameter
170  *             entity, the name will begin with '%', and if it is the
171  *             external DTD subset, it will be "[dtd]".
172  * @exception SAXException The application may raise an exception.
173  * @see #endEntity
174  * @see org.xml.sax.ext.DeclHandler#internalEntityDecl
175  * @see org.xml.sax.ext.DeclHandler#externalEntityDecl
176  */
177 public abstract void startEntity (String name)
178     throws SAXException;
179
180

```

```
181 /**
182  * Report the end of an entity.
183  *
184  * @param name The name of the entity that is ending.
185  * @exception SAXException The application may raise an exception.
186  * @see #startEntity
187  */
188 public abstract void endEntity (String name)
189     throws SAXException;
190
191
192 /**
193  * Report the start of a CDATA section.
194  *
195  * <p>The contents of the CDATA section will be reported through
196  * the regular {@link org.xml.sax.ContentHandler#characters
197  * characters} event; this event is intended only to report
198  * the boundary.</p>
199  *
200  * @exception SAXException The application may raise an exception.
201  * @see #endCDATA
202  */
203 public abstract void startCDATA ()
204     throws SAXException;
205
206
207 /**
208  * Report the end of a CDATA section.
209  *
210  * @exception SAXException The application may raise an exception.
211  * @see #startCDATA
212  */
213 public abstract void endCDATA ()
214     throws SAXException;
215
216
217 /**
218  * Report an XML comment anywhere in the document.
219  *
220  * <p>This callback will be used for comments inside or outside the
221  * document element, including comments in the external DTD
222  * subset (if read). Comments in the DTD must be properly
223  * nested inside start/endDTD and start/endEntity events (if
224  * used).</p>
225  *
226  * @param ch An array holding the characters in the comment.
227  * @param start The starting position in the array.
228  * @param length The number of characters to use from the array.
229  * @exception SAXException The application may raise an exception.
230  */
231 public abstract void comment (char ch[], int start, int length)
232     throws SAXException;
233
```

```
234 }
235
236 // end of LexicalHandler.java
```

## 48.7 Locator2.java

Secuencia 501: org/xml/sax/ext/Locator2.java

```
1  /*
2  * Copyright (c) 2004, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // Locator2.java - extended Locator
27 // http://www.saxproject.org
28 // Public Domain: no warranty.
29 // $Id: Locator2.java,v 1.2 2004/11/03 22:49:08 jsuttor Exp $
30
31 package org.xml.sax.ext;
32
33 import org.xml.sax.Locator;
34
35
36 /**
37 * SAX2 extension to augment the entity information provided
38 * though a {@link Locator}.
39 * If an implementation supports this extension, the Locator
40 * provided in {@link org.xml.sax.ContentHandler#setDocumentLocator
41 * ContentHandler.setDocumentLocator() } will implement this
42 * interface, and the
43 * <em>http://xml.org/sax/features/use-locator2</em> feature
44 * flag will have the value <em>true</em>.
45 *
46 * <blockquote>
```

```

47 * <em>This module, both source code and documentation, is in the
48 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
49 * </blockquote>
50 *
51 * <p>XMLReader implementations are not required to support this
52 * information, and it is not part of core-only SAX2 distributions.</p>
53 *
54 * @since SAX 2.0 (extensions 1.1 alpha)
55 * @author David Brownell
56 */
57 public interface Locator2 extends Locator
58 {
59     /**
60      * Returns the version of XML used for the entity. This will
61      * normally be the identifier from the current entity's
62      * <em><?xml&nbsp;version='...'&nbsp;...?&gt;</em> declaration,
63      * or be defaulted by the parser.
64      *
65      * @return Identifier for the XML version being used to interpret
66      * the entity's text, or null if that information is not yet
67      * available in the current parsing state.
68      */
69     public String getXMLVersion ();
70
71     /**
72      * Returns the name of the character encoding for the entity.
73      * If the encoding was declared externally (for example, in a MIME
74      * Content-Type header), that will be the name returned. Else if there
75      * was an <em><?xml&nbsp;...encoding='...'&gt;</em> declaration at
76      * the start of the document, that encoding name will be returned.
77      * Otherwise the encoding will be inferred (normally to be UTF-8, or
78      * some UTF-16 variant), and that inferred name will be returned.
79      *
80      * <p>When an {@link org.xml.sax.InputSource InputSource} is used
81      * to provide an entity's character stream, this method returns the
82      * encoding provided in that input stream.
83      *
84      * <p>Note that some recent W3C specifications require that text
85      * in some encodings be normalized, using Unicode Normalization
86      * Form C, before processing. Such normalization must be performed
87      * by applications, and would normally be triggered based on the
88      * value returned by this method.
89      *
90      * <p>Encoding names may be those used by the underlying JVM,
91      * and comparisons should be case-insensitive.
92      *
93      * @return Name of the character encoding being used to interpret
94      * the entity's text, or null if this was not provided for a *
95      * character stream passed through an InputSource or is otherwise
96      * not yet available in the current parsing state.
97      */
98     public String getEncoding ();
99 }

```

## 48.8 Locator2Impl.java

Secuencia 502: org/xml/sax/ext/Locator2Impl.java

```
1  /*
2  * Copyright (c) 2004, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // Locator2Impl.java - extended LocatorImpl
27 // http://www.saxproject.org
28 // Public Domain: no warranty.
29 // $Id: Locator2Impl.java,v 1.2 2004/11/03 22:49:08 jsuttor Exp $
30
31 package org.xml.sax.ext;
32
33 import org.xml.sax.Locator;
34 import org.xml.sax.helpers.LocatorImpl;
35
36
37 /**
38 * SAX2 extension helper for holding additional Entity information,
39 * implementing the {@link Locator2} interface.
40 *
41 * <blockquote>
42 * <em>This module, both source code and documentation, is in the
43 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
44 * </blockquote>
45 *
46 * <p> This is not part of core-only SAX2 distributions.</p>
47 *
48 * @since SAX 2.0.2
49 * @author David Brownell
50 */
```

```

51 public class Locator2Impl extends LocatorImpl implements Locator2
52 {
53     private String      encoding;
54     private String      version;
55
56
57     /**
58      * Construct a new, empty Locator2Impl object.
59      * This will not normally be useful, since the main purpose
60      * of this class is to make a snapshot of an existing Locator.
61      */
62     public Locator2Impl () { }
63
64     /**
65      * Copy an existing Locator or Locator2 object.
66      * If the object implements Locator2, values of the
67      * <em>encoding</em> and <em>version</em> strings are copied,
68      * otherwise they set to <em>null</em>.
69      *
70      * @param locator The existing Locator object.
71      */
72     public Locator2Impl (Locator locator)
73     {
74         super (locator);
75         if (locator instanceof Locator2) {
76             Locator2 l2 = (Locator2) locator;
77
78             version = l2.getXMLVersion ();
79             encoding = l2.getEncoding ();
80         }
81     }
82
83     ////////////////////////////////////////////////////
84     // Locator2 method implementations
85     ////////////////////////////////////////////////////
86
87     /**
88      * Returns the current value of the version property.
89      *
90      * @see #setXMLVersion
91      */
92     public String getXMLVersion ()
93     { return version; }
94
95     /**
96      * Returns the current value of the encoding property.
97      *
98      * @see #setEncoding
99      */
100    public String getEncoding ()
101    { return encoding; }
102
103

```

```

104 ///////////////////////////////////////////////////
105 // Setters
106 ///////////////////////////////////////////////////
107
108 /**
109  * Assigns the current value of the version property.
110  *
111  * @param version the new "version" value
112  * @see #getXMLVersion
113  */
114 public void setXMLVersion (String version)
115     { this.version = version; }
116
117 /**
118  * Assigns the current value of the encoding property.
119  *
120  * @param encoding the new "encoding" value
121  * @see #getEncoding
122  */
123 public void setEncoding (String encoding)
124     { this.encoding = encoding; }
125 }

```

## 49 Sax Helpers (org.xml.sax.helpers package)

### 49.1 AttributeListImpl.java

Secuencia 503: org/xml/sax/helpers/AttributeListImpl.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25

```



```
26 // SAX default implementation for AttributeList.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: AttributeListImpl.java,v 1.2 2004/11/03 22:53:08 jsuttor Exp $
30
31 package org.xml.sax.helpers;
32
33 import org.xml.sax.AttributeList;
34
35 import java.util.Vector;
36
37
38 /**
39  * Default implementation for AttributeList.
40  *
41  * <blockquote>
42  * <em>This module, both source code and documentation, is in the
43  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
44  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
45  * for further information.
46  * </blockquote>
47  *
48  * <p>AttributeList implements the deprecated SAX1 {@link
49  * org.xml.sax.AttributeList AttributeList} interface, and has been
50  * replaced by the new SAX2 {@link org.xml.sax.helpers.AttributesImpl
51  * AttributesImpl} interface.</p>
52  *
53  * <p>This class provides a convenience implementation of the SAX
54  * {@link org.xml.sax.AttributeList AttributeList} interface. This
55  * implementation is useful both for SAX parser writers, who can use
56  * it to provide attributes to the application, and for SAX application
57  * writers, who can use it to create a persistent copy of an element's
58  * attribute specifications:</p>
59  *
60  * <pre>
61  * private AttributeList myatts;
62  *
63  * public void startElement (String name, AttributeList atts)
64  * {
65  *     // create a persistent copy of the attribute list
66  *     // for use outside this method
67  *     myatts = new AttributeListImpl(atts);
68  *     [...]
69  * }
70  * </pre>
71  *
72  * <p>Please note that SAX parsers are not required to use this
73  * class to provide an implementation of AttributeList; it is
74  * supplied only as an optional convenience. In particular,
75  * parser writers are encouraged to invent more efficient
76  * implementations.</p>
77  *
78  * @deprecated This class implements a deprecated interface,
```

```

79  *      {@link org.xml.sax.AttributeList AttributeList};
80  *      that interface has been replaced by
81  *      {@link org.xml.sax.Attributes Attributes},
82  *      which is implemented in the
83  *      {@link org.xml.sax.helpers.AttributesImpl
84  *      AttributesImpl} helper class.
85  * @since SAX 1.0
86  * @author David Megginson
87  * @see org.xml.sax.AttributeList
88  * @see org.xml.sax.DocumentHandler#startElement
89  */
90 public class AttributeListImpl implements AttributeList
91 {
92
93     /**
94      * Create an empty attribute list.
95      *
96      * <p>This constructor is most useful for parser writers, who
97      * will use it to create a single, reusable attribute list that
98      * can be reset with the clear method between elements.</p>
99      *
100     * @see #addAttribute
101     * @see #clear
102     */
103     public AttributeListImpl ()
104     {
105     }
106
107
108     /**
109      * Construct a persistent copy of an existing attribute list.
110      *
111      * <p>This constructor is most useful for application writers,
112      * who will use it to create a persistent copy of an existing
113      * attribute list.</p>
114      *
115      * @param atts The attribute list to copy
116      * @see org.xml.sax.DocumentHandler#startElement
117      */
118     public AttributeListImpl (AttributeList atts)
119     {
120         setAttributeList(atts);
121     }
122
123
124
125     //////////////////////////////////////
126     // Methods specific to this class.
127     //////////////////////////////////////
128
129
130     /**
131      * Set the attribute list, discarding previous contents.

```

```
132     *
133     * <p>This method allows an application writer to reuse an
134     * attribute list easily.</p>
135     *
136     * @param atts The attribute list to copy.
137     */
138     public void setAttributeList (AttributeList atts)
139     {
140         int count = atts.getLength();
141
142         clear();
143
144         for (int i = 0; i < count; i++) {
145             addAttribute(atts.getName(i), atts.getType(i), atts.getValue(i));
146         }
147     }
148
149
150     /**
151     * Add an attribute to an attribute list.
152     *
153     * <p>This method is provided for SAX parser writers, to allow them
154     * to build up an attribute list incrementally before delivering
155     * it to the application.</p>
156     *
157     * @param name The attribute name.
158     * @param type The attribute type ("NM_TOKEN" for an enumeration).
159     * @param value The attribute value (must not be null).
160     * @see #removeAttribute
161     * @see org.xml.sax.DocumentHandler#startElement
162     */
163     public void addAttribute (String name, String type, String value)
164     {
165         names.addElement(name);
166         types.addElement(type);
167         values.addElement(value);
168     }
169
170
171     /**
172     * Remove an attribute from the list.
173     *
174     * <p>SAX application writers can use this method to filter an
175     * attribute out of an AttributeList. Note that invoking this
176     * method will change the length of the attribute list and
177     * some of the attribute's indices.</p>
178     *
179     * <p>If the requested attribute is not in the list, this is
180     * a no-op.</p>
181     *
182     * @param name The attribute name.
183     * @see #addAttribute
184     */
```

```

185 public void removeAttribute (String name)
186 {
187     int i = names.indexOf(name);
188
189     if (i >= 0) {
190         names.removeElementAt(i);
191         types.removeElementAt(i);
192         values.removeElementAt(i);
193     }
194 }
195
196
197 /**
198  * Clear the attribute list.
199  *
200  * <p>SAX parser writers can use this method to reset the attribute
201  * list between DocumentHandler.startElement events. Normally,
202  * it will make sense to reuse the same AttributeListImpl object
203  * rather than allocating a new one each time.</p>
204  *
205  * @see org.xml.sax.DocumentHandler#startElement
206  */
207 public void clear ()
208 {
209     names.removeAllElements();
210     types.removeAllElements();
211     values.removeAllElements();
212 }
213
214
215
216 ///////////////////////////////////////////////////
217 // Implementation of org.xml.sax.AttributeList
218 ///////////////////////////////////////////////////
219
220
221 /**
222  * Return the number of attributes in the list.
223  *
224  * @return The number of attributes in the list.
225  * @see org.xml.sax.AttributeList#getLength
226  */
227 public int getLength ()
228 {
229     return names.size();
230 }
231
232
233 /**
234  * Get the name of an attribute (by position).
235  *
236  * @param i The position of the attribute in the list.
237  * @return The attribute name as a string, or null if there

```

```
238     *         is no attribute at that position.
239     * @see org.xml.sax.AttributeList#getName(int)
240     */
241     public String getName (int i)
242     {
243         if (i < 0) {
244             return null;
245         }
246         try {
247             return (String)names.elementAt(i);
248         } catch (ArrayIndexOutOfBoundsException e) {
249             return null;
250         }
251     }
252
253
254     /**
255     * Get the type of an attribute (by position).
256     *
257     * @param i The position of the attribute in the list.
258     * @return The attribute type as a string ("NM_TOKEN" for an
259     *         enumeration, and "CDATA" if no declaration was
260     *         read), or null if there is no attribute at
261     *         that position.
262     * @see org.xml.sax.AttributeList#getType(int)
263     */
264     public String getType (int i)
265     {
266         if (i < 0) {
267             return null;
268         }
269         try {
270             return (String)types.elementAt(i);
271         } catch (ArrayIndexOutOfBoundsException e) {
272             return null;
273         }
274     }
275
276
277     /**
278     * Get the value of an attribute (by position).
279     *
280     * @param i The position of the attribute in the list.
281     * @return The attribute value as a string, or null if
282     *         there is no attribute at that position.
283     * @see org.xml.sax.AttributeList#getValue(int)
284     */
285     public String getValue (int i)
286     {
287         if (i < 0) {
288             return null;
289         }
290         try {
```

```

291         return (String)values.elementAt(i);
292     } catch (ArrayIndexOutOfBoundsException e) {
293         return null;
294     }
295 }
296
297
298 /**
299  * Get the type of an attribute (by name).
300  *
301  * @param name The attribute name.
302  * @return The attribute type as a string ("NMTOKEN" for an
303  *         enumeration, and "CDATA" if no declaration was
304  *         read).
305  * @see org.xml.sax.AttributeList#getType(java.lang.String)
306  */
307 public String getType (String name)
308 {
309     return getType(names.indexOf(name));
310 }
311
312
313 /**
314  * Get the value of an attribute (by name).
315  *
316  * @param name The attribute name.
317  * @see org.xml.sax.AttributeList#getValue(java.lang.String)
318  */
319 public String getValue (String name)
320 {
321     return getValue(names.indexOf(name));
322 }
323
324
325
326 ///////////////////////////////////////////////////
327 // Internal state.
328 ///////////////////////////////////////////////////
329
330 Vector names = new Vector();
331 Vector types = new Vector();
332 Vector values = new Vector();
333
334 }
335
336 // end of AttributeListImpl.java

```

## 49.2 AttributesImpl.java

Secuencia 504: org/xml/sax/helpers/AttributesImpl.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

```

```
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // AttributesImpl.java - default implementation of Attributes.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the public domain.
30 // $Id: AttributesImpl.java,v 1.2 2004/11/03 22:53:08 jsuttor Exp $
31
32 package org.xml.sax.helpers;
33
34 import org.xml.sax.Attributes;
35
36
37 /**
38  * Default implementation of the Attributes interface.
39  *
40  * <blockquote>
41  * <em>This module, both source code and documentation, is in the
42  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
43  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
44  * for further information.
45  * </blockquote>
46  *
47  * <p>This class provides a default implementation of the SAX2
48  * {@link org.xml.sax.Attributes Attributes} interface, with the
49  * addition of manipulators so that the list can be modified or
50  * reused.</p>
51  *
52  * <p>There are two typical uses of this class:</p>
53  *
54  * <ol>
55  * <li>to take a persistent snapshot of an Attributes object
56  * in a {@link org.xml.sax.ContentHandler#startElement startElement} event; or</li>
```





```
110     * Return the number of attributes in the list.
111     *
112     * @return The number of attributes in the list.
113     * @see org.xml.sax.Attributes#getLength
114     */
115     public int getLength ()
116     {
117         return length;
118     }
119
120
121     /**
122     * Return an attribute's Namespace URI.
123     *
124     * @param index The attribute's index (zero-based).
125     * @return The Namespace URI, the empty string if none is
126     *         available, or null if the index is out of range.
127     * @see org.xml.sax.Attributes#getURI
128     */
129     public String getURI (int index)
130     {
131         if (index >= 0 && index < length) {
132             return data[index*5];
133         } else {
134             return null;
135         }
136     }
137
138
139     /**
140     * Return an attribute's local name.
141     *
142     * @param index The attribute's index (zero-based).
143     * @return The attribute's local name, the empty string if
144     *         none is available, or null if the index if out of range.
145     * @see org.xml.sax.Attributes#getLocalName
146     */
147     public String getLocalName (int index)
148     {
149         if (index >= 0 && index < length) {
150             return data[index*5+1];
151         } else {
152             return null;
153         }
154     }
155
156
157     /**
158     * Return an attribute's qualified (prefixed) name.
159     *
160     * @param index The attribute's index (zero-based).
161     * @return The attribute's qualified name, the empty string if
162     *         none is available, or null if the index is out of bounds.
```

```
163     * @see org.xml.sax.Attributes#getQName
164     */
165     public String getQName (int index)
166     {
167         if (index >= 0 && index < length) {
168             return data[index*5+2];
169         } else {
170             return null;
171         }
172     }
173
174
175     /**
176     * Return an attribute's type by index.
177     *
178     * @param index The attribute's index (zero-based).
179     * @return The attribute's type, "CDATA" if the type is unknown, or null
180     *         if the index is out of bounds.
181     * @see org.xml.sax.Attributes#getType(int)
182     */
183     public String getType (int index)
184     {
185         if (index >= 0 && index < length) {
186             return data[index*5+3];
187         } else {
188             return null;
189         }
190     }
191
192
193     /**
194     * Return an attribute's value by index.
195     *
196     * @param index The attribute's index (zero-based).
197     * @return The attribute's value or null if the index is out of bounds.
198     * @see org.xml.sax.Attributes#getValue(int)
199     */
200     public String getValue (int index)
201     {
202         if (index >= 0 && index < length) {
203             return data[index*5+4];
204         } else {
205             return null;
206         }
207     }
208
209
210     /**
211     * Look up an attribute's index by Namespace name.
212     *
213     * <p>In many cases, it will be more efficient to look up the name once and
214     * use the index query methods rather than using the name query methods
215     * repeatedly.</p>
```

```

216     *
217     * @param uri The attribute's Namespace URI, or the empty
218     *           string if none is available.
219     * @param localName The attribute's local name.
220     * @return The attribute's index, or -1 if none matches.
221     * @see org.xml.sax.Attributes#getIndex(java.lang.String,java.lang.String)
222     */
223     public int getIndex (String uri, String localName)
224     {
225         int max = length * 5;
226         for (int i = 0; i < max; i += 5) {
227             if (data[i].equals(uri) && data[i+1].equals(localName)) {
228                 return i / 5;
229             }
230         }
231         return -1;
232     }
233
234
235     /**
236     * Look up an attribute's index by qualified (prefixed) name.
237     *
238     * @param qName The qualified name.
239     * @return The attribute's index, or -1 if none matches.
240     * @see org.xml.sax.Attributes#getIndex(java.lang.String)
241     */
242     public int getIndex (String qName)
243     {
244         int max = length * 5;
245         for (int i = 0; i < max; i += 5) {
246             if (data[i+2].equals(qName)) {
247                 return i / 5;
248             }
249         }
250         return -1;
251     }
252
253
254     /**
255     * Look up an attribute's type by Namespace-qualified name.
256     *
257     * @param uri The Namespace URI, or the empty string for a name
258     *           with no explicit Namespace URI.
259     * @param localName The local name.
260     * @return The attribute's type, or null if there is no
261     *         matching attribute.
262     * @see org.xml.sax.Attributes#getType(java.lang.String,java.lang.String)
263     */
264     public String getType (String uri, String localName)
265     {
266         int max = length * 5;
267         for (int i = 0; i < max; i += 5) {
268             if (data[i].equals(uri) && data[i+1].equals(localName)) {

```

```
269         return data[i+3];
270     }
271 }
272 return null;
273 }
274
275
276 /**
277  * Look up an attribute's type by qualified (prefixed) name.
278  *
279  * @param qName The qualified name.
280  * @return The attribute's type, or null if there is no
281  *         matching attribute.
282  * @see org.xml.sax.Attributes#getType(java.lang.String)
283  */
284 public String getType (String qName)
285 {
286     int max = length * 5;
287     for (int i = 0; i < max; i += 5) {
288         if (data[i+2].equals(qName)) {
289             return data[i+3];
290         }
291     }
292     return null;
293 }
294
295
296 /**
297  * Look up an attribute's value by Namespace-qualified name.
298  *
299  * @param uri The Namespace URI, or the empty string for a name
300  *         with no explicit Namespace URI.
301  * @param localName The local name.
302  * @return The attribute's value, or null if there is no
303  *         matching attribute.
304  * @see org.xml.sax.Attributes#getValue(java.lang.String,java.lang.String)
305  */
306 public String getValue (String uri, String localName)
307 {
308     int max = length * 5;
309     for (int i = 0; i < max; i += 5) {
310         if (data[i].equals(uri) && data[i+1].equals(localName)) {
311             return data[i+4];
312         }
313     }
314     return null;
315 }
316
317
318 /**
319  * Look up an attribute's value by qualified (prefixed) name.
320  *
321  * @param qName The qualified name.
```

```

322     * @return The attribute's value, or null if there is no
323     *         matching attribute.
324     * @see org.xml.sax.Attributes#getValue(java.lang.String)
325     */
326     public String getValue (String qName)
327     {
328         int max = length * 5;
329         for (int i = 0; i < max; i += 5) {
330             if (data[i+2].equals(qName)) {
331                 return data[i+4];
332             }
333         }
334         return null;
335     }
336
337
338
339     //////////////////////////////////////
340     // Manipulators.
341     //////////////////////////////////////
342
343
344     /**
345     * Clear the attribute list for reuse.
346     *
347     * <p>Note that little memory is freed by this call:
348     * the current array is kept so it can be
349     * reused.</p>
350     */
351     public void clear ()
352     {
353         if (data != null) {
354             for (int i = 0; i < (length * 5); i++)
355                 data [i] = null;
356         }
357         length = 0;
358     }
359
360
361     /**
362     * Copy an entire Attributes object.
363     *
364     * <p>It may be more efficient to reuse an existing object
365     * rather than constantly allocating new ones.</p>
366     *
367     * @param atts The attributes to copy.
368     */
369     public void setAttributes (Attributes atts)
370     {
371         clear();
372         length = atts.getLength();
373         if (length > 0) {
374             data = new String[length*5];

```

```

375         for (int i = 0; i < length; i++) {
376             data[i*5] = atts.getURI(i);
377             data[i*5+1] = atts.getLocalName(i);
378             data[i*5+2] = atts.getQName(i);
379             data[i*5+3] = atts.getType(i);
380             data[i*5+4] = atts.getValue(i);
381         }
382     }
383 }
384
385
386 /**
387  * Add an attribute to the end of the list.
388  *
389  * <p>For the sake of speed, this method does no checking
390  * to see if the attribute is already in the list: that is
391  * the responsibility of the application.</p>
392  *
393  * @param uri The Namespace URI, or the empty string if
394  *            none is available or Namespace processing is not
395  *            being performed.
396  * @param localName The local name, or the empty string if
397  *            Namespace processing is not being performed.
398  * @param qName The qualified (prefixed) name, or the empty string
399  *            if qualified names are not available.
400  * @param type The attribute type as a string.
401  * @param value The attribute value.
402  */
403 public void addAttribute (String uri, String localName, String qName,
404                          String type, String value)
405 {
406     ensureCapacity(length+1);
407     data[length*5] = uri;
408     data[length*5+1] = localName;
409     data[length*5+2] = qName;
410     data[length*5+3] = type;
411     data[length*5+4] = value;
412     length++;
413 }
414
415
416 /**
417  * Set an attribute in the list.
418  *
419  * <p>For the sake of speed, this method does no checking
420  * for name conflicts or well-formedness: such checks are the
421  * responsibility of the application.</p>
422  *
423  * @param index The index of the attribute (zero-based).
424  * @param uri The Namespace URI, or the empty string if
425  *            none is available or Namespace processing is not
426  *            being performed.
427  * @param localName The local name, or the empty string if

```

```

428     *      Namespace processing is not being performed.
429     * @param qName The qualified name, or the empty string
430     *      if qualified names are not available.
431     * @param type The attribute type as a string.
432     * @param value The attribute value.
433     * @exception java.lang.ArrayIndexOutOfBoundsException When the
434     *      supplied index does not point to an attribute
435     *      in the list.
436     */
437 public void setAttribute (int index, String uri, String localName,
438                          String qName, String type, String value)
439 {
440     if (index >= 0 && index < length) {
441         data[index*5] = uri;
442         data[index*5+1] = localName;
443         data[index*5+2] = qName;
444         data[index*5+3] = type;
445         data[index*5+4] = value;
446     } else {
447         badIndex(index);
448     }
449 }
450
451
452 /**
453  * Remove an attribute from the list.
454  *
455  * @param index The index of the attribute (zero-based).
456  * @exception java.lang.ArrayIndexOutOfBoundsException When the
457  *      supplied index does not point to an attribute
458  *      in the list.
459  */
460 public void removeAttribute (int index)
461 {
462     if (index >= 0 && index < length) {
463         if (index < length - 1) {
464             System.arraycopy(data, (index+1)*5, data, index*5,
465                             (length-index-1)*5);
466         }
467         index = (length - 1) * 5;
468         data [index++] = null;
469         data [index++] = null;
470         data [index++] = null;
471         data [index++] = null;
472         data [index] = null;
473         length--;
474     } else {
475         badIndex(index);
476     }
477 }
478
479
480 /**

```

```
481     * Set the Namespace URI of a specific attribute.
482     *
483     * @param index The index of the attribute (zero-based).
484     * @param uri The attribute's Namespace URI, or the empty
485     *           string for none.
486     * @exception java.lang.ArrayIndexOutOfBoundsException When the
487     *           supplied index does not point to an attribute
488     *           in the list.
489     */
490     public void setURI (int index, String uri)
491     {
492         if (index >= 0 && index < length) {
493             data[index*5] = uri;
494         } else {
495             badIndex(index);
496         }
497     }
498
499     /**
500     * Set the local name of a specific attribute.
501     *
502     *
503     * @param index The index of the attribute (zero-based).
504     * @param localName The attribute's local name, or the empty
505     *           string for none.
506     * @exception java.lang.ArrayIndexOutOfBoundsException When the
507     *           supplied index does not point to an attribute
508     *           in the list.
509     */
510     public void setLocalName (int index, String localName)
511     {
512         if (index >= 0 && index < length) {
513             data[index*5+1] = localName;
514         } else {
515             badIndex(index);
516         }
517     }
518
519     /**
520     * Set the qualified name of a specific attribute.
521     *
522     *
523     * @param index The index of the attribute (zero-based).
524     * @param qName The attribute's qualified name, or the empty
525     *           string for none.
526     * @exception java.lang.ArrayIndexOutOfBoundsException When the
527     *           supplied index does not point to an attribute
528     *           in the list.
529     */
530     public void setQName (int index, String qName)
531     {
532         if (index >= 0 && index < length) {
533             data[index*5+2] = qName;
```



```

534     } else {
535         badIndex(index);
536     }
537 }
538
539
540 /**
541  * Set the type of a specific attribute.
542  *
543  * @param index The index of the attribute (zero-based).
544  * @param type The attribute's type.
545  * @exception java.lang.ArrayIndexOutOfBoundsException When the
546  *             supplied index does not point to an attribute
547  *             in the list.
548  */
549 public void setType (int index, String type)
550 {
551     if (index >= 0 && index < length) {
552         data[index*5+3] = type;
553     } else {
554         badIndex(index);
555     }
556 }
557
558
559 /**
560  * Set the value of a specific attribute.
561  *
562  * @param index The index of the attribute (zero-based).
563  * @param value The attribute's value.
564  * @exception java.lang.ArrayIndexOutOfBoundsException When the
565  *             supplied index does not point to an attribute
566  *             in the list.
567  */
568 public void setValue (int index, String value)
569 {
570     if (index >= 0 && index < length) {
571         data[index*5+4] = value;
572     } else {
573         badIndex(index);
574     }
575 }
576
577
578
579 ////////////////////////////////////////////////////
580 // Internal methods.
581 ////////////////////////////////////////////////////
582
583
584 /**
585  * Ensure the internal array's capacity.
586  *

```

```

587     * @param n The minimum number of attributes that the array must
588     *         be able to hold.
589     */
590     private void ensureCapacity (int n)    {
591         if (n <= 0) {
592             return;
593         }
594         int max;
595         if (data == null || data.length == 0) {
596             max = 25;
597         }
598         else if (data.length >= n * 5) {
599             return;
600         }
601         else {
602             max = data.length;
603         }
604         while (max < n * 5) {
605             max *= 2;
606         }
607
608         String newData[] = new String[max];
609         if (length > 0) {
610             System.arraycopy(data, 0, newData, 0, length*5);
611         }
612         data = newData;
613     }
614
615     /**
616     * Report a bad array index in a manipulator.
617     *
618     * @param index The index to report.
619     * @exception java.lang.ArrayIndexOutOfBoundsException Always.
620     */
621
622     private void badIndex (int index)
623         throws ArrayIndexOutOfBoundsException
624     {
625         String msg =
626             "Attempt to modify attribute at illegal index: " + index;
627         throw new ArrayIndexOutOfBoundsException(msg);
628     }
629
630
631
632     //////////////////////////////////////
633     // Internal state.
634     //////////////////////////////////////
635
636     int length;
637     String data [];
638
639 }

```

```
640
641 // end of AttributesImpl.java
```

### 49.3 DefaultHandler.java

Secuencia 505: org/xml/sax/helpers/DefaultHandler.java

```
1  /*
2  * Copyright (c) 2000, 2006, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // DefaultHandler.java - default implementation of the core handlers.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the public domain.
30 // $Id: DefaultHandler.java,v 1.3 2006/04/13 02:06:32 jeffsuttor Exp $
31
32 package org.xml.sax.helpers;
33
34 import java.io.IOException;
35
36 import org.xml.sax.InputSource;
37 import org.xml.sax.Locator;
38 import org.xml.sax.Attributes;
39 import org.xml.sax.EntityResolver;
40 import org.xml.sax.DTDHandler;
41 import org.xml.sax.ContentHandler;
42 import org.xml.sax.ErrorHandler;
43 import org.xml.sax.SAXException;
44 import org.xml.sax.SAXParseException;
45
46
47 /**
```

```

48  * Default base class for SAX2 event handlers.
49  *
50  * <blockquote>
51  * <em>This module, both source code and documentation, is in the
52  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
53  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
54  * for further information.
55  * </blockquote>
56  *
57  * <p>This class is available as a convenience base class for SAX2
58  * applications: it provides default implementations for all of the
59  * callbacks in the four core SAX2 handler classes:</p>
60  *
61  * <ul>
62  * <li>{@link org.xml.sax.EntityResolver EntityResolver}</li>
63  * <li>{@link org.xml.sax.DTDHandler DTDHandler}</li>
64  * <li>{@link org.xml.sax.ContentHandler ContentHandler}</li>
65  * <li>{@link org.xml.sax.ErrorHandler ErrorHandler}</li>
66  * </ul>
67  *
68  * <p>Application writers can extend this class when they need to
69  * implement only part of an interface; parser writers can
70  * instantiate this class to provide default handlers when the
71  * application has not supplied its own.</p>
72  *
73  * <p>This class replaces the deprecated SAX1
74  * {@link org.xml.sax.HandlerBase HandlerBase} class.</p>
75  *
76  * @since SAX 2.0
77  * @author David Megginson,
78  * @see org.xml.sax.EntityResolver
79  * @see org.xml.sax.DTDHandler
80  * @see org.xml.sax.ContentHandler
81  * @see org.xml.sax.ErrorHandler
82  */
83 public class DefaultHandler
84     implements EntityResolver, DTDHandler, ContentHandler, ErrorHandler
85 {
86
87
88     ////////////////////////////////////////////////////
89     // Default implementation of the EntityResolver interface.
90     ////////////////////////////////////////////////////
91
92     /**
93      * Resolve an external entity.
94      *
95      * <p>Always return null, so that the parser will use the system
96      * identifier provided in the XML document. This method implements
97      * the SAX default behaviour: application writers can override it
98      * in a subclass to do special translations such as catalog lookups
99      * or URI redirection.</p>
100     */

```

```

101     * @param publicId The public identifier, or null if none is
102     *                 available.
103     * @param systemId The system identifier provided in the XML
104     *                 document.
105     * @return The new input source, or null to require the
106     *         default behaviour.
107     * @exception java.io.IOException If there is an error setting
108     *         up the new input source.
109     * @exception org.xml.sax.SAXException Any SAX exception, possibly
110     *         wrapping another exception.
111     * @see org.xml.sax.EntityResolver#resolveEntity
112     */
113     public InputSource resolveEntity (String publicId, String systemId)
114         throws IOException, SAXException
115     {
116         return null;
117     }
118
119
120
121     //////////////////////////////////////
122     // Default implementation of DTDHandler interface.
123     //////////////////////////////////////
124
125
126     /**
127     * Receive notification of a notation declaration.
128     *
129     * <p>By default, do nothing. Application writers may override this
130     * method in a subclass if they wish to keep track of the notations
131     * declared in a document.</p>
132     *
133     * @param name The notation name.
134     * @param publicId The notation public identifier, or null if not
135     *                 available.
136     * @param systemId The notation system identifier.
137     * @exception org.xml.sax.SAXException Any SAX exception, possibly
138     *         wrapping another exception.
139     * @see org.xml.sax.DTDHandler#notationDecl
140     */
141     public void notationDecl (String name, String publicId, String systemId)
142         throws SAXException
143     {
144         // no op
145     }
146
147
148     /**
149     * Receive notification of an unparsed entity declaration.
150     *
151     * <p>By default, do nothing. Application writers may override this
152     * method in a subclass to keep track of the unparsed entities
153     * declared in a document.</p>

```

```

154      *
155      * @param name The entity name.
156      * @param publicId The entity public identifier, or null if not
157      *         available.
158      * @param systemId The entity system identifier.
159      * @param notationName The name of the associated notation.
160      * @exception org.xml.sax.SAXException Any SAX exception, possibly
161      *         wrapping another exception.
162      * @see org.xml.sax.DTDHandler#unparsedEntityDecl
163      */
164      public void unparsedEntityDecl (String name, String publicId,
165                                     String systemId, String notationName)
166          throws SAXException
167      {
168          // no op
169      }
170
171
172
173      //////////////////////////////////////
174      // Default implementation of ContentHandler interface.
175      //////////////////////////////////////
176
177
178      /**
179       * Receive a Locator object for document events.
180       *
181       * <p>By default, do nothing. Application writers may override this
182       * method in a subclass if they wish to store the locator for use
183       * with other document events.</p>
184       *
185       * @param locator A locator for all SAX document events.
186       * @see org.xml.sax.ContentHandler#setDocumentLocator
187       * @see org.xml.sax.Locator
188       */
189      public void setDocumentLocator (Locator locator)
190      {
191          // no op
192      }
193
194
195      /**
196       * Receive notification of the beginning of the document.
197       *
198       * <p>By default, do nothing. Application writers may override this
199       * method in a subclass to take specific actions at the beginning
200       * of a document (such as allocating the root node of a tree or
201       * creating an output file).</p>
202       *
203       * @exception org.xml.sax.SAXException Any SAX exception, possibly
204       *         wrapping another exception.
205       * @see org.xml.sax.ContentHandler#startDocument
206       */

```

```
207 public void startDocument ()
208     throws SAXException
209 {
210     // no op
211 }
212
213
214 /**
215  * Receive notification of the end of the document.
216  *
217  * <p>By default, do nothing. Application writers may override this
218  * method in a subclass to take specific actions at the end
219  * of a document (such as finalising a tree or closing an output
220  * file).</p>
221  *
222  * @exception org.xml.sax.SAXException Any SAX exception, possibly
223  *         wrapping another exception.
224  * @see org.xml.sax.ContentHandler#endDocument
225  */
226 public void endDocument ()
227     throws SAXException
228 {
229     // no op
230 }
231
232
233 /**
234  * Receive notification of the start of a Namespace mapping.
235  *
236  * <p>By default, do nothing. Application writers may override this
237  * method in a subclass to take specific actions at the start of
238  * each Namespace prefix scope (such as storing the prefix mapping).</p>
239  *
240  * @param prefix The Namespace prefix being declared.
241  * @param uri The Namespace URI mapped to the prefix.
242  * @exception org.xml.sax.SAXException Any SAX exception, possibly
243  *         wrapping another exception.
244  * @see org.xml.sax.ContentHandler#startPrefixMapping
245  */
246 public void startPrefixMapping (String prefix, String uri)
247     throws SAXException
248 {
249     // no op
250 }
251
252
253 /**
254  * Receive notification of the end of a Namespace mapping.
255  *
256  * <p>By default, do nothing. Application writers may override this
257  * method in a subclass to take specific actions at the end of
258  * each prefix mapping.</p>
259  *
```

```
260     * @param prefix The Namespace prefix being declared.
261     * @exception org.xml.sax.SAXException Any SAX exception, possibly
262     *         wrapping another exception.
263     * @see org.xml.sax.ContentHandler#endPrefixMapping
264     */
265     public void endPrefixMapping (String prefix)
266         throws SAXException
267     {
268         // no op
269     }
270
271
272     /**
273     * Receive notification of the start of an element.
274     *
275     * <p>By default, do nothing. Application writers may override this
276     * method in a subclass to take specific actions at the start of
277     * each element (such as allocating a new tree node or writing
278     * output to a file).</p>
279     *
280     * @param uri The Namespace URI, or the empty string if the
281     *         element has no Namespace URI or if Namespace
282     *         processing is not being performed.
283     * @param localName The local name (without prefix), or the
284     *         empty string if Namespace processing is not being
285     *         performed.
286     * @param qName The qualified name (with prefix), or the
287     *         empty string if qualified names are not available.
288     * @param attributes The attributes attached to the element. If
289     *         there are no attributes, it shall be an empty
290     *         Attributes object.
291     * @exception org.xml.sax.SAXException Any SAX exception, possibly
292     *         wrapping another exception.
293     * @see org.xml.sax.ContentHandler#startElement
294     */
295     public void startElement (String uri, String localName,
296                             String qName, Attributes attributes)
297         throws SAXException
298     {
299         // no op
300     }
301
302
303     /**
304     * Receive notification of the end of an element.
305     *
306     * <p>By default, do nothing. Application writers may override this
307     * method in a subclass to take specific actions at the end of
308     * each element (such as finalising a tree node or writing
309     * output to a file).</p>
310     *
311     * @param uri The Namespace URI, or the empty string if the
312     *         element has no Namespace URI or if Namespace
```



```

313     *      processing is not being performed.
314     * @param localName The local name (without prefix), or the
315     *      empty string if Namespace processing is not being
316     *      performed.
317     * @param qName The qualified name (with prefix), or the
318     *      empty string if qualified names are not available.
319     * @exception org.xml.sax.SAXException Any SAX exception, possibly
320     *      wrapping another exception.
321     * @see org.xml.sax.ContentHandler#endElement
322     */
323     public void endElement (String uri, String localName, String qName)
324         throws SAXException
325     {
326         // no op
327     }
328
329
330     /**
331     * Receive notification of character data inside an element.
332     *
333     * <p>By default, do nothing. Application writers may override this
334     * method to take specific actions for each chunk of character data
335     * (such as adding the data to a node or buffer, or printing it to
336     * a file).</p>
337     *
338     * @param ch The characters.
339     * @param start The start position in the character array.
340     * @param length The number of characters to use from the
341     *      character array.
342     * @exception org.xml.sax.SAXException Any SAX exception, possibly
343     *      wrapping another exception.
344     * @see org.xml.sax.ContentHandler#characters
345     */
346     public void characters (char ch[], int start, int length)
347         throws SAXException
348     {
349         // no op
350     }
351
352
353     /**
354     * Receive notification of ignorable whitespace in element content.
355     *
356     * <p>By default, do nothing. Application writers may override this
357     * method to take specific actions for each chunk of ignorable
358     * whitespace (such as adding data to a node or buffer, or printing
359     * it to a file).</p>
360     *
361     * @param ch The whitespace characters.
362     * @param start The start position in the character array.
363     * @param length The number of characters to use from the
364     *      character array.
365     * @exception org.xml.sax.SAXException Any SAX exception, possibly

```

```
366     *           wrapping another exception.
367     * @see org.xml.sax.ContentHandler#ignorableWhitespace
368     */
369     public void ignorableWhitespace (char ch[], int start, int length)
370         throws SAXException
371     {
372         // no op
373     }
374
375
376     /**
377     * Receive notification of a processing instruction.
378     *
379     * <p>By default, do nothing. Application writers may override this
380     * method in a subclass to take specific actions for each
381     * processing instruction, such as setting status variables or
382     * invoking other methods.</p>
383     *
384     * @param target The processing instruction target.
385     * @param data The processing instruction data, or null if
386     *             none is supplied.
387     * @exception org.xml.sax.SAXException Any SAX exception, possibly
388     *            wrapping another exception.
389     * @see org.xml.sax.ContentHandler#processingInstruction
390     */
391     public void processingInstruction (String target, String data)
392         throws SAXException
393     {
394         // no op
395     }
396
397
398     /**
399     * Receive notification of a skipped entity.
400     *
401     * <p>By default, do nothing. Application writers may override this
402     * method in a subclass to take specific actions for each
403     * processing instruction, such as setting status variables or
404     * invoking other methods.</p>
405     *
406     * @param name The name of the skipped entity.
407     * @exception org.xml.sax.SAXException Any SAX exception, possibly
408     *            wrapping another exception.
409     * @see org.xml.sax.ContentHandler#processingInstruction
410     */
411     public void skippedEntity (String name)
412         throws SAXException
413     {
414         // no op
415     }
416
417
418
```

```
419 //////////////////////////////////////////////////
420 // Default implementation of the ErrorHandler interface.
421 //////////////////////////////////////////////////
422
423
424 /**
425  * Receive notification of a parser warning.
426  *
427  * <p>The default implementation does nothing. Application writers
428  * may override this method in a subclass to take specific actions
429  * for each warning, such as inserting the message in a log file or
430  * printing it to the console.</p>
431  *
432  * @param e The warning information encoded as an exception.
433  * @exception org.xml.sax.SAXException Any SAX exception, possibly
434  *         wrapping another exception.
435  * @see org.xml.sax.ErrorHandler#warning
436  * @see org.xml.sax.SAXParseException
437  */
438 public void warning (SAXParseException e)
439     throws SAXException
440 {
441     // no op
442 }
443
444
445 /**
446  * Receive notification of a recoverable parser error.
447  *
448  * <p>The default implementation does nothing. Application writers
449  * may override this method in a subclass to take specific actions
450  * for each error, such as inserting the message in a log file or
451  * printing it to the console.</p>
452  *
453  * @param e The error information encoded as an exception.
454  * @exception org.xml.sax.SAXException Any SAX exception, possibly
455  *         wrapping another exception.
456  * @see org.xml.sax.ErrorHandler#warning
457  * @see org.xml.sax.SAXParseException
458  */
459 public void error (SAXParseException e)
460     throws SAXException
461 {
462     // no op
463 }
464
465
466 /**
467  * Report a fatal XML parsing error.
468  *
469  * <p>The default implementation throws a SAXParseException.
470  * Application writers may override this method in a subclass if
471  * they need to take specific actions for each fatal error (such as
```

```

472     * collecting all of the errors into a single report): in any case,
473     * the application must stop all regular processing when this
474     * method is invoked, since the document is no longer reliable, and
475     * the parser may no longer report parsing events.</p>
476     *
477     * @param e The error information encoded as an exception.
478     * @exception org.xml.sax.SAXException Any SAX exception, possibly
479     *         wrapping another exception.
480     * @see org.xml.sax.ErrorHandler#fatalError
481     * @see org.xml.sax.SAXParseException
482     */
483     public void fatalError (SAXParseException e)
484         throws SAXException
485     {
486         throw e;
487     }
488 }
489 }
490
491 // end of DefaultHandler.java

```

#### 49.4 LocatorImpl.java

Secuencia 506: org/xml/sax/helpers/LocatorImpl.java

```

1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX default implementation for Locator.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: LocatorImpl.java,v 1.2 2004/11/03 22:53:09 jsuttor Exp $

```

```
30
31 package org.xml.sax.helpers;
32
33 import org.xml.sax Locator;
34
35
36 /**
37  * Provide an optional convenience implementation of Locator.
38  *
39  * <blockquote>
40  * <em>This module, both source code and documentation, is in the
41  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
42  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
43  * for further information.
44  * </blockquote>
45  *
46  * <p>This class is available mainly for application writers, who
47  * can use it to make a persistent snapshot of a locator at any
48  * point during a document parse:</p>
49  *
50  * <pre>
51  * Locator locator;
52  * Locator startloc;
53  *
54  * public void setLocator (Locator locator)
55  * {
56  *     // note the locator
57  *     this.locator = locator;
58  * }
59  *
60  * public void startDocument ()
61  * {
62  *     // save the location of the start of the document
63  *     // for future use.
64  *     Locator startloc = new LocatorImpl(locator);
65  * }
66  *</pre>
67  *
68  * <p>Normally, parser writers will not use this class, since it
69  * is more efficient to provide location information only when
70  * requested, rather than constantly updating a Locator object.</p>
71  *
72  * @since SAX 1.0
73  * @author David Megginson
74  * @see org.xml.sax.Locator Locator
75  */
76 public class LocatorImpl implements Locator
77 {
78
79
80     /**
81      * Zero-argument constructor.
82      *
```

```

83      * <p>This will not normally be useful, since the main purpose
84      * of this class is to make a snapshot of an existing Locator.</p>
85      */
86      public LocatorImpl ()
87      {
88      }
89
90
91      /**
92       * Copy constructor.
93       *
94       * <p>Create a persistent copy of the current state of a locator.
95       * When the original locator changes, this copy will still keep
96       * the original values (and it can be used outside the scope of
97       * DocumentHandler methods).</p>
98       *
99       * @param locator The locator to copy.
100      */
101      public LocatorImpl (Locator locator)
102      {
103          setPublicId(locator.getPublicId());
104          setSystemId(locator.getSystemId());
105          setLineNumber(locator.getLineNumber());
106          setColumnNumber(locator.getColumnNumber());
107      }
108
109
110
111      //////////////////////////////////////
112      // Implementation of org.xml.sax.Locator
113      //////////////////////////////////////
114
115
116      /**
117       * Return the saved public identifier.
118       *
119       * @return The public identifier as a string, or null if none
120       *         is available.
121       * @see org.xml.sax.Locator#getPublicId
122       * @see #setPublicId
123       */
124      public String getPublicId ()
125      {
126          return publicId;
127      }
128
129
130      /**
131       * Return the saved system identifier.
132       *
133       * @return The system identifier as a string, or null if none
134       *         is available.
135       * @see org.xml.sax.Locator#getSystemId

```

```
136     * @see #setSystemId
137     */
138     public String getSystemId ()
139     {
140         return systemId;
141     }
142
143
144     /**
145     * Return the saved line number (1-based).
146     *
147     * @return The line number as an integer, or -1 if none is available.
148     * @see org.xml.sax.Locator#getLineNumber
149     * @see #setLineNumber
150     */
151     public int getLineNumber ()
152     {
153         return lineNumber;
154     }
155
156
157     /**
158     * Return the saved column number (1-based).
159     *
160     * @return The column number as an integer, or -1 if none is available.
161     * @see org.xml.sax.Locator#getColumnNumber
162     * @see #setColumnNumber
163     */
164     public int getColumnNumber ()
165     {
166         return columnNumber;
167     }
168
169
170
171     //////////////////////////////////////
172     // Setters for the properties (not in org.xml.sax.Locator)
173     //////////////////////////////////////
174
175
176     /**
177     * Set the public identifier for this locator.
178     *
179     * @param publicId The new public identifier, or null
180     *                 if none is available.
181     * @see #getPublicId
182     */
183     public void setPublicId (String publicId)
184     {
185         this.publicId = publicId;
186     }
187
188
```

```

189  /**
190   * Set the system identifier for this locator.
191   *
192   * @param systemId The new system identifier, or null
193   *     if none is available.
194   * @see #getSystemId
195   */
196  public void setSystemId (String systemId)
197  {
198      this.systemId = systemId;
199  }
200
201
202  /**
203   * Set the line number for this locator (1-based).
204   *
205   * @param lineNumber The line number, or -1 if none is available.
206   * @see #getLineNumber
207   */
208  public void setLineNumber (int lineNumber)
209  {
210      this.lineNumber = lineNumber;
211  }
212
213
214  /**
215   * Set the column number for this locator (1-based).
216   *
217   * @param columnNumber The column number, or -1 if none is available.
218   * @see #getColumnNumber
219   */
220  public void setColumnNumber (int columnNumber)
221  {
222      this.columnNumber = columnNumber;
223  }
224
225
226
227  //////////////////////////////////////
228  // Internal state.
229  //////////////////////////////////////
230
231  private String publicId;
232  private String systemId;
233  private int lineNumber;
234  private int columnNumber;
235
236  }
237
238  // end of LocatorImpl.java

```

## 49.5 NamespaceSupport.java



Secuencia 507: org/xml/sax/helpers/NamespaceSupport.java

```
1  /*
2  * Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // NamespaceSupport.java - generic Namespace support for SAX.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // This class is in the Public Domain. NO WARRANTY!
30 // $Id: NamespaceSupport.java,v 1.5 2004/11/03 22:53:09 jsuttor Exp $
31
32 package org.xml.sax.helpers;
33
34 import java.util.ArrayList;
35 import java.util.Collections;
36 import java.util.EmptyStackException;
37 import java.util.Enumeration;
38 import java.util.HashMap;
39 import java.util.List;
40 import java.util.Map;
41
42
43 /**
44 * Encapsulate Namespace logic for use by applications using SAX,
45 * or internally by SAX drivers.
46 *
47 * <blockquote>
48 * <em>This module, both source code and documentation, is in the
49 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
50 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
51 * for further information.
52 * </blockquote>
```

```

53  *
54  * <p>This class encapsulates the logic of Namespace processing: it
55  * tracks the declarations currently in force for each context and
56  * automatically processes qualified XML names into their Namespace
57  * parts; it can also be used in reverse for generating XML qnames
58  * from Namespaces.</p>
59  *
60  * <p>Namespace support objects are reusable, but the reset method
61  * must be invoked between each session.</p>
62  *
63  * <p>Here is a simple session:</p>
64  *
65  * <pre>
66  * String parts[] = new String[3];
67  * NamespaceSupport support = new NamespaceSupport();
68  *
69  * support.pushContext();
70  * support.declarePrefix("", "http://www.w3.org/1999/xhtml");
71  * support.declarePrefix("dc", "http://www.purl.org/dc#");
72  *
73  * parts = support.processName("p", parts, false);
74  * System.out.println("Namespace URI: " + parts[0]);
75  * System.out.println("Local name: " + parts[1]);
76  * System.out.println("Raw name: " + parts[2]);
77  *
78  * parts = support.processName("dc:title", parts, false);
79  * System.out.println("Namespace URI: " + parts[0]);
80  * System.out.println("Local name: " + parts[1]);
81  * System.out.println("Raw name: " + parts[2]);
82  *
83  * support.popContext();
84  * </pre>
85  *
86  * <p>Note that this class is optimized for the use case where most
87  * elements do not contain Namespace declarations: if the same
88  * prefix/URI mapping is repeated for each context (for example), this
89  * class will be somewhat less efficient.</p>
90  *
91  * <p>Although SAX drivers (parsers) may choose to use this class to
92  * implement namespace handling, they are not required to do so.
93  * Applications must track namespace information themselves if they
94  * want to use namespace information.
95  *
96  * @since SAX 2.0
97  * @author David Megginson
98  */
99  public class NamespaceSupport
100  {
101
102
103      //////////////////////////////////////
104      // Constants.
105      //////////////////////////////////////

```

```

106
107
108 /**
109  * The XML Namespace URI as a constant.
110  * The value is <code>http://www.w3.org/XML/1998/namespace</code>
111  * as defined in the "Namespaces in XML" * recommendation.
112  *
113  * <p>This is the Namespace URI that is automatically mapped
114  * to the "xml" prefix.</p>
115  */
116 public final static String XMLNS =
117     "http://www.w3.org/XML/1998/namespace";
118
119
120 /**
121  * The namespace declaration URI as a constant.
122  * The value is <code>http://www.w3.org/xmlns/2000/</code>, as defined
123  * in a backwards-incompatible erratum to the "Namespaces in XML"
124  * recommendation. Because that erratum postdated SAX2, SAX2 defaults
125  * to the original recommendation, and does not normally use this URI.
126  *
127  *
128  * <p>This is the Namespace URI that is optionally applied to
129  * <em>xmlns</em> and <em>xmlns:*</em> attributes, which are used to
130  * declare namespaces. </p>
131  *
132  * @since SAX 2.1alpha
133  * @see #setNamespaceDeclUris
134  * @see #isNamespaceDeclUris
135  */
136 public final static String NSDECL =
137     "http://www.w3.org/xmlns/2000/";
138
139
140 /**
141  * An empty enumeration.
142  */
143 private final static Enumeration EMPTY_ENUMERATION =
144     Collections.enumeration(new ArrayList<String>());
145
146
147 //////////////////////////////////////
148 // Constructor.
149 //////////////////////////////////////
150
151
152 /**
153  * Create a new Namespace support object.
154  */
155 public NamespaceSupport ()
156 {
157     reset();
158 }

```

```

159
160
161
162 //////////////////////////////////////////////////
163 // Context management.
164 //////////////////////////////////////////////////
165
166
167 /**
168  * Reset this Namespace support object for reuse.
169  *
170  * <p>It is necessary to invoke this method before reusing the
171  * Namespace support object for a new session. If namespace
172  * declaration URIs are to be supported, that flag must also
173  * be set to a non-default value.
174  * </p>
175  *
176  * @see #setNamespaceDeclUri
177  */
178 public void reset ()
179 {
180     contexts = new Context[32];
181     namespaceDeclUri = false;
182     contextPos = 0;
183     contexts[contextPos] = currentContext = new Context();
184     currentContext.declarePrefix("xml", XMLNS);
185 }
186
187
188 /**
189  * Start a new Namespace context.
190  * The new context will automatically inherit
191  * the declarations of its parent context, but it will also keep
192  * track of which declarations were made within this context.
193  *
194  * <p>Event callback code should start a new context once per element.
195  * This means being ready to call this in either of two places.
196  * For elements that don't include namespace declarations, the
197  * <em>ContentHandler.startElement()</em> callback is the right place.
198  * For elements with such a declaration, it'd done in the first
199  * <em>ContentHandler.startPrefixMapping()</em> callback.
200  * A boolean flag can be used to
201  * track whether a context has been started yet. When either of
202  * those methods is called, it checks the flag to see if a new context
203  * needs to be started. If so, it starts the context and sets the
204  * flag. After <em>ContentHandler.startElement()</em>
205  * does that, it always clears the flag.
206  *
207  * <p>Normally, SAX drivers would push a new context at the beginning
208  * of each XML element. Then they perform a first pass over the
209  * attributes to process all namespace declarations, making
210  * <em>ContentHandler.startPrefixMapping()</em> callbacks.
211  * Then a second pass is made, to determine the namespace-qualified

```

```

212     * names for all attributes and for the element name.
213     * Finally all the information for the
214     * <em>ContentHandler.startElement()</em> callback is available,
215     * so it can then be made.
216     *
217     * <p>The Namespace support object always starts with a base context
218     * already in force: in this context, only the "xml" prefix is
219     * declared.</p>
220     *
221     * @see org.xml.sax.ContentHandler
222     * @see #popContext
223     */
224     public void pushContext ()
225     {
226         int max = contexts.length;
227
228         contextPos++;
229
230         // Extend the array if necessary
231         if (contextPos >= max) {
232             Context newContexts[] = new Context[max*2];
233             System.arraycopy(contexts, 0, newContexts, 0, max);
234             max *= 2;
235             contexts = newContexts;
236         }
237
238         // Allocate the context if necessary.
239         currentContext = contexts[contextPos];
240         if (currentContext == null) {
241             contexts[contextPos] = currentContext = new Context();
242         }
243
244         // Set the parent, if any.
245         if (contextPos > 0) {
246             currentContext.setParent(contexts[contextPos - 1]);
247         }
248     }
249
250
251     /**
252     * Revert to the previous Namespace context.
253     *
254     * <p>Normally, you should pop the context at the end of each
255     * XML element. After popping the context, all Namespace prefix
256     * mappings that were previously in force are restored.</p>
257     *
258     * <p>You must not attempt to declare additional Namespace
259     * prefixes after popping a context, unless you push another
260     * context first.</p>
261     *
262     * @see #pushContext
263     */
264     public void popContext ()

```

```

265     {
266         contexts[contextPos].clear();
267         contextPos--;
268         if (contextPos < 0) {
269             throw new EmptyStackException();
270         }
271         currentContext = contexts[contextPos];
272     }
273
274
275
276     //////////////////////////////////////
277     // Operations within a context.
278     //////////////////////////////////////
279
280
281     /**
282      * Declare a Namespace prefix. All prefixes must be declared
283      * before they are referenced. For example, a SAX driver (parser)
284      * would scan an element's attributes
285      * in two passes: first for namespace declarations,
286      * then a second pass using {@link #processName processName()} to
287      * interpret prefixes against (potentially redefined) prefixes.
288      *
289      * <p>This method declares a prefix in the current Namespace
290      * context; the prefix will remain in force until this context
291      * is popped, unless it is shadowed in a descendant context.</p>
292      *
293      * <p>To declare the default element Namespace, use the empty string as
294      * the prefix.</p>
295      *
296      * <p>Note that there is an asymmetry in this library: {@link
297      * #getPrefix getPrefix} will not return the "" prefix,
298      * even if you have declared a default element namespace.
299      * To check for a default namespace,
300      * you have to look it up explicitly using {@link #getURI getURI}.
301      * This asymmetry exists to make it easier to look up prefixes
302      * for attribute names, where the default prefix is not allowed.</p>
303      *
304      * @param prefix The prefix to declare, or the empty string to
305      * indicate the default element namespace. This may never have
306      * the value "xml" or "xmlns".
307      * @param uri The Namespace URI to associate with the prefix.
308      * @return true if the prefix was legal, false otherwise
309      *
310      * @see #processName
311      * @see #getURI
312      * @see #getPrefix
313      */
314     public boolean declarePrefix (String prefix, String uri)
315     {
316         if (prefix.equals("xml") || prefix.equals("xmlns")) {
317             return false;

```

```

318     } else {
319         currentContext.declarePrefix(prefix, uri);
320         return true;
321     }
322 }
323
324
325 /**
326  * Process a raw XML qualified name, after all declarations in the
327  * current context have been handled by {@link #declarePrefix
328  * declarePrefix()}.
329  *
330  * <p>This method processes a raw XML qualified name in the
331  * current context by removing the prefix and looking it up among
332  * the prefixes currently declared. The return value will be the
333  * array supplied by the caller, filled in as follows:</p>
334  *
335  * <dl>
336  * <dt>parts[0]</dt>
337  * <dd>The Namespace URI, or an empty string if none is
338  * in use.</dd>
339  * <dt>parts[1]</dt>
340  * <dd>The local name (without prefix).</dd>
341  * <dt>parts[2]</dt>
342  * <dd>The original raw name.</dd>
343  * </dl>
344  *
345  * <p>All of the strings in the array will be internalized. If
346  * the raw name has a prefix that has not been declared, then
347  * the return value will be null.</p>
348  *
349  * <p>Note that attribute names are processed differently than
350  * element names: an unprefixed element name will receive the
351  * default Namespace (if any), while an unprefixed attribute name
352  * will not.</p>
353  *
354  * @param qName The XML qualified name to be processed.
355  * @param parts An array supplied by the caller, capable of
356  * holding at least three members.
357  * @param isAttribute A flag indicating whether this is an
358  * attribute name (true) or an element name (false).
359  * @return The supplied array holding three internalized strings
360  * representing the Namespace URI (or empty string), the
361  * local name, and the XML qualified name; or null if there
362  * is an undeclared prefix.
363  * @see #declarePrefix
364  * @see java.lang.String#intern */
365 public String [] processName (String qName, String parts[],
366                             boolean isAttribute)
367 {
368     String myParts[] = currentContext.processName(qName, isAttribute);
369     if (myParts == null) {
370         return null;

```

```
371     } else {
372         parts[0] = myParts[0];
373         parts[1] = myParts[1];
374         parts[2] = myParts[2];
375         return parts;
376     }
377 }
378
379
380 /**
381  * Look up a prefix and get the currently-mapped Namespace URI.
382  *
383  * <p>This method looks up the prefix in the current context.
384  * Use the empty string ("") for the default Namespace.</p>
385  *
386  * @param prefix The prefix to look up.
387  * @return The associated Namespace URI, or null if the prefix
388  *         is undeclared in this context.
389  * @see #getPrefix
390  * @see #getPrefixes
391  */
392 public String getURI (String prefix)
393 {
394     return currentContext.getURI(prefix);
395 }
396
397
398 /**
399  * Return an enumeration of all prefixes whose declarations are
400  * active in the current context.
401  * This includes declarations from parent contexts that have
402  * not been overridden.
403  *
404  * <p><strong>Note:</strong> if there is a default prefix, it will not be
405  * returned in this enumeration; check for the default prefix
406  * using the {@link #getURI getURI} with an argument of "".</p>
407  *
408  * @return An enumeration of prefixes (never empty).
409  * @see #getDeclaredPrefixes
410  * @see #getURI
411  */
412 public Enumeration getPrefixes ()
413 {
414     return currentContext.getPrefixes();
415 }
416
417
418 /**
419  * Return one of the prefixes mapped to a Namespace URI.
420  *
421  * <p>If more than one prefix is currently mapped to the same
422  * URI, this method will make an arbitrary selection; if you
423  * want all of the prefixes, use the {@link #getPrefixes}
```



```

424     * method instead.</p>
425     *
426     * <p><strong>Note:</strong> this will never return the empty (default) prefix;
427     * to check for a default prefix, use the {@link #getURI getURI}
428     * method with an argument of "".</p>
429     *
430     * @param uri the namespace URI
431     * @return one of the prefixes currently mapped to the URI supplied,
432     *         or null if none is mapped or if the URI is assigned to
433     *         the default namespace
434     * @see #getPrefixes(java.lang.String)
435     * @see #getURI
436     */
437     public String getPrefix (String uri)
438     {
439         return currentContext.getPrefix(uri);
440     }
441
442
443     /**
444     * Return an enumeration of all prefixes for a given URI whose
445     * declarations are active in the current context.
446     * This includes declarations from parent contexts that have
447     * not been overridden.
448     *
449     * <p>This method returns prefixes mapped to a specific Namespace
450     * URI. The xml: prefix will be included. If you want only one
451     * prefix that's mapped to the Namespace URI, and you don't care
452     * which one you get, use the {@link #getPrefix getPrefix}
453     * method instead.</p>
454     *
455     * <p><strong>Note:</strong> the empty (default) prefix is <em>never</em> included
456     * in this enumeration; to check for the presence of a default
457     * Namespace, use the {@link #getURI getURI} method with an
458     * argument of "".</p>
459     *
460     * @param uri The Namespace URI.
461     * @return An enumeration of prefixes (never empty).
462     * @see #getPrefix
463     * @see #getDeclaredPrefixes
464     * @see #getURI
465     */
466     public Enumeration getPrefixes (String uri)
467     {
468         List<String> prefixes = new ArrayList<>();
469         Enumeration allPrefixes = getPrefixes();
470         while (allPrefixes.hasMoreElements()) {
471             String prefix = (String)allPrefixes.nextElement();
472             if (uri.equals(getURI(prefix))) {
473                 prefixes.add(prefix);
474             }
475         }
476         return Collections.enumeration(prefixes);

```

```
477     }
478
479
480     /**
481      * Return an enumeration of all prefixes declared in this context.
482      *
483      * <p>The empty (default) prefix will be included in this
484      * enumeration; note that this behaviour differs from that of
485      * {@link #getPrefix} and {@link #getPrefixes}.</p>
486      *
487      * @return An enumeration of all prefixes declared in this
488      *         context.
489      * @see #getPrefixes
490      * @see #getURI
491      */
492     public Enumeration getDeclaredPrefixes ()
493     {
494         return currentContext.getDeclaredPrefixes();
495     }
496
497     /**
498      * Controls whether namespace declaration attributes are placed
499      * into the {@link #NSDECL NSDECL} namespace
500      * by {@link #processName processName()}. This may only be
501      * changed before any contexts have been pushed.
502      *
503      * @since SAX 2.1alpha
504      *
505      * @exception IllegalStateException when attempting to set this
506      *         after any context has been pushed.
507      */
508     public void setNamespaceDeclUris (boolean value)
509     {
510         if (contextPos != 0)
511             throw new IllegalStateException ();
512         if (value == namespaceDeclUris)
513             return;
514         namespaceDeclUris = value;
515         if (value)
516             currentContext.declarePrefix ("xmlns", NSDECL);
517         else {
518             contexts[contextPos] = currentContext = new Context();
519             currentContext.declarePrefix("xml", XMLNS);
520         }
521     }
522
523     /**
524      * Returns true if namespace declaration attributes are placed into
525      * a namespace. This behavior is not the default.
526      *
527      * @since SAX 2.1alpha
528      */
529     public boolean isNamespaceDeclUris ()
```

```

530     { return namespaceDeclUris; }
531
532
533
534     //////////////////////////////////////
535     // Internal state.
536     //////////////////////////////////////
537
538     private Context contexts[];
539     private Context currentContext;
540     private int contextPos;
541     private boolean namespaceDeclUris;
542
543
544     //////////////////////////////////////
545     // Internal classes.
546     //////////////////////////////////////
547
548     /**
549      * Internal class for a single Namespace context.
550      *
551      * <p>This module caches and reuses Namespace contexts,
552      * so the number allocated
553      * will be equal to the element depth of the document, not to the total
554      * number of elements (i.e. 5-10 rather than tens of thousands).
555      * Also, data structures used to represent contexts are shared when
556      * possible (child contexts without declarations) to further reduce
557      * the amount of memory that's consumed.
558      * </p>
559      */
560     final class Context {
561
562         /**
563          * Create the root-level Namespace context.
564          */
565         Context ()
566         {
567             copyTables();
568         }
569
570
571         /**
572          * (Re)set the parent of this Namespace context.
573          * The context must either have been freshly constructed,
574          * or must have been cleared.
575          *
576          * @param context The parent Namespace context object.
577          */
578         void setParent (Context parent)
579         {
580             this.parent = parent;
581             declarations = null;
582             prefixTable = parent.prefixTable;

```

```

583         uriTable = parent.uriTable;
584         elementNameTable = parent.elementNameTable;
585         attributeNameTable = parent.attributeNameTable;
586         defaultNS = parent.defaultNS;
587         declSeen = false;
588     }
589
590     /**
591     * Makes associated state become collectible,
592     * invalidating this context.
593     * {@link #setParent} must be called before
594     * this context may be used again.
595     */
596     void clear ()
597     {
598         parent = null;
599         prefixTable = null;
600         uriTable = null;
601         elementNameTable = null;
602         attributeNameTable = null;
603         defaultNS = null;
604     }
605
606
607     /**
608     * Declare a Namespace prefix for this context.
609     *
610     * @param prefix The prefix to declare.
611     * @param uri The associated Namespace URI.
612     * @see org.xml.sax.helpers.NamespaceSupport#declarePrefix
613     */
614     void declarePrefix (String prefix, String uri)
615     {
616         // Lazy processing...
617         if (!declsOK)
618             throw new IllegalStateException (
619                 "can't declare any more prefixes in this context");
620         if (!declSeen) {
621             copyTables();
622         }
623         if (declarations == null) {
624             declarations = new ArrayList<>();
625         }
626
627         prefix = prefix.intern();
628         uri = uri.intern();
629         if ("".equals(prefix)) {
630             if ("".equals(uri)) {
631                 defaultNS = null;
632             } else {
633                 defaultNS = uri;
634             }
635         } else {

```

```

636         prefixTable.put(prefix, uri);
637         uriTable.put(uri, prefix); // may wipe out another prefix
638     }
639     declarations.add(prefix);
640 }
641
642
643 /**
644  * Process an XML qualified name in this context.
645  *
646  * @param qName The XML qualified name.
647  * @param isAttribute true if this is an attribute name.
648  * @return An array of three strings containing the
649  *         URI part (or empty string), the local part,
650  *         and the raw name, all internalized, or null
651  *         if there is an undeclared prefix.
652  * @see org.xml.sax.helpers.NamespaceSupport#processName
653  */
654 String [] processName (String qName, boolean isAttribute)
655 {
656     String name[];
657     Map<String, String[]> table;
658
659     // Select the appropriate table.
660     if (isAttribute) {
661         table = attributeNameTable;
662     } else {
663         table = elementNameTable;
664     }
665
666     // Start by looking in the cache, and
667     // return immediately if the name
668     // is already known in this content
669     name = (String[])table.get(qName);
670     if (name != null) {
671         return name;
672     }
673
674     // We haven't seen this name in this
675     // context before. Maybe in the parent
676     // context, but we can't assume prefix
677     // bindings are the same.
678     name = new String[3];
679     name[2] = qName.intern();
680     int index = qName.indexOf(':');
681
682
683     // No prefix.
684     if (index == -1) {
685         if (isAttribute) {
686             if (qName == "xmlns" && namespaceDeclUris)
687                 name[0] = NSDECL;
688             else

```

```

689         name[0] = "";
690     } else if (defaultNS == null) {
691         name[0] = "";
692     } else {
693         name[0] = defaultNS;
694     }
695     name[1] = name[2];
696 }
697
698 // Prefix
699 else {
700     String prefix = qName.substring(0, index);
701     String local = qName.substring(index+1);
702     String uri;
703     if ("".equals(prefix)) {
704         uri = defaultNS;
705     } else {
706         uri = (String)prefixTable.get(prefix);
707     }
708     if (uri == null
709         || (!isAttribute && "xmlns".equals (prefix))) {
710         return null;
711     }
712     name[0] = uri;
713     name[1] = local.intern();
714 }
715
716 // Save in the cache for future use.
717 // (Could be shared with parent context...)
718 table.put(name[2], name);
719 return name;
720 }
721
722
723 /**
724  * Look up the URI associated with a prefix in this context.
725  *
726  * @param prefix The prefix to look up.
727  * @return The associated Namespace URI, or null if none is
728  *         declared.
729  * @see org.xml.sax.helpers.NamespaceSupport#getURI
730  */
731 String getURI (String prefix)
732 {
733     if ("".equals(prefix)) {
734         return defaultNS;
735     } else if (prefixTable == null) {
736         return null;
737     } else {
738         return (String)prefixTable.get(prefix);
739     }
740 }
741

```

```
742
743 /**
744  * Look up one of the prefixes associated with a URI in this context.
745  *
746  * <p>Since many prefixes may be mapped to the same URI,
747  * the return value may be unreliable.</p>
748  *
749  * @param uri The URI to look up.
750  * @return The associated prefix, or null if none is declared.
751  * @see org.xml.sax.helpers.NamespaceSupport#getPrefix
752  */
753 String getPrefix (String uri)
754 {
755     if (uriTable == null) {
756         return null;
757     } else {
758         return (String)uriTable.get(uri);
759     }
760 }
761
762
763 /**
764  * Return an enumeration of prefixes declared in this context.
765  *
766  * @return An enumeration of prefixes (possibly empty).
767  * @see org.xml.sax.helpers.NamespaceSupport#getDeclaredPrefixes
768  */
769 Enumeration getDeclaredPrefixes ()
770 {
771     if (declarations == null) {
772         return EMPTY_ENUMERATION;
773     } else {
774         return Collections.enumeration(declarations);
775     }
776 }
777
778 /**
779  * Return an enumeration of all prefixes currently in force.
780  *
781  * <p>The default prefix, if in force, is <em>not</em>
782  * returned, and will have to be checked for separately.</p>
783  *
784  * @return An enumeration of prefixes (never empty).
785  * @see org.xml.sax.helpers.NamespaceSupport#getPrefixes
786  */
787 Enumeration getPrefixes ()
788 {
789     if (prefixTable == null) {
790         return EMPTY_ENUMERATION;
791     } else {
792         return Collections.enumeration(prefixTable.keySet());
793     }
794 }
```

```

795
796
797
798 ///////////////////////////////////////////////////
799 // Internal methods.
800 ///////////////////////////////////////////////////
801
802
803 /**
804  * Copy on write for the internal tables in this context.
805  *
806  * <p>This class is optimized for the normal case where most
807  * elements do not contain Namespace declarations.</p>
808  */
809 private void copyTables ()
810 {
811     if (prefixTable != null) {
812         prefixTable = new HashMap<>(prefixTable);
813     } else {
814         prefixTable = new HashMap<>();
815     }
816     if (uriTable != null) {
817         uriTable = new HashMap<>(uriTable);
818     } else {
819         uriTable = new HashMap<>();
820     }
821     elementNameTable = new HashMap<>();
822     attributeNameTable = new HashMap<>();
823     declSeen = true;
824 }
825
826
827
828 ///////////////////////////////////////////////////
829 // Protected state.
830 ///////////////////////////////////////////////////
831
832 Map<String, String> prefixTable;
833 Map<String, String> uriTable;
834 Map<String, String[]> elementNameTable;
835 Map<String, String[]> attributeNameTable;
836 String defaultNS = null;
837
838
839
840 ///////////////////////////////////////////////////
841 // Internal state.
842 ///////////////////////////////////////////////////
843
844 private List<String> declarations = null;
845 private boolean declSeen = false;
846 private Context parent = null;
847 }

```



```
848 }
849
850 // end of NamespaceSupport.java
```

## 49.6 NewInstance.java

Secuencia 508: org/xml/sax/helpers/NewInstance.java

```
1  /*
2  * Copyright (c) 2001, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // NewInstance.java - create a new instance of a class by name.
27 // http://www.saxproject.org
28 // Written by Edwin Goetz, edwingo@apache.org
29 // and by David Brownell, dbrownell@users.sourceforge.net
30 // NO WARRANTY! This class is in the Public Domain.
31 // $Id: NewInstance.java,v 1.2 2005/06/10 03:50:50 jeffsuttor Exp $
32
33 package org.xml.sax.helpers;
34
35 import java.lang.reflect.Method;
36 import java.lang.reflect.InvocationTargetException;
37
38 /**
39 * Create a new instance of a class by name.
40 *
41 * <blockquote>
42 * <em>This module, both source code and documentation, is in the
43 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
44 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
45 * for further information.
46 * </blockquote>
```

```

47  *
48  * <p>This class contains a static method for creating an instance of a
49  * class from an explicit class name. It tries to use the thread's context
50  * ClassLoader if possible and falls back to using
51  * Class.forName(String).</p>
52  *
53  * <p>This code is designed to compile and run on JDK version 1.1 and later
54  * including versions of Java 2.</p>
55  *
56  * @author Edwin Goei, David Brownell
57  * @version 2.0.1 (sax2r2)
58  */
59  class NewInstance {
60      private static final String DEFAULT_PACKAGE = "com.sun.org.apache.xerces.internal";
61      /**
62       * Creates a new instance of the specified class name
63       *
64       * Package private so this code is not exposed at the API level.
65       */
66      static Object newInstance (ClassLoader classLoader, String className)
67          throws ClassNotFoundException, IllegalAccessException,
68              InstantiationException
69      {
70          // make sure we have access to restricted packages
71          boolean internal = false;
72          if (System.getSecurityManager() != null) {
73              if (className != null && className.startsWith(DEFAULT_PACKAGE)) {
74                  internal = true;
75              }
76          }
77
78          Class driverClass;
79          if (classLoader == null || internal) {
80              driverClass = Class.forName(className);
81          } else {
82              driverClass = classLoader.loadClass(className);
83          }
84          return driverClass.newInstance();
85      }
86  }
87 }

```

## 49.7 ParserAdapter.java

Secuencia 509: org/xml/sax/helpers/ParserAdapter.java

```

1  /*
2  * Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *

```

```
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // ParserAdapter.java - adapt a SAX1 Parser to a SAX2 XMLReader.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the public domain.
30 // $Id: ParserAdapter.java,v 1.3 2004/11/03 22:53:09 jsuttor Exp $
31
32 package org.xml.sax.helpers;
33
34 import java.io.IOException;
35 import java.util.Enumeration;
36 import java.util.Vector;
37
38 import org.xml.sax.Parser;      // deprecated
39 import org.xml.sax.InputSource;
40 import org.xml.sax Locator;
41 import org.xml.sax.AttributeList; // deprecated
42 import org.xml.sax.EntityResolver;
43 import org.xml.sax.DTDHandler;
44 import org.xml.sax.DocumentHandler; // deprecated
45 import org.xml.sax.ErrorHandler;
46 import org.xml.sax.SAXException;
47 import org.xml.sax.SAXParseException;
48
49 import org.xml.sax.XMLReader;
50 import org.xml.sax.Attributes;
51 import org.xml.sax.ContentHandler;
52 import org.xml.sax.SAXNotRecognizedException;
53 import org.xml.sax.SAXNotSupportedException;
54
55
56 /**
57  * Adapt a SAX1 Parser as a SAX2 XMLReader.
58  *
59  * <blockquote>
60  * <em>This module, both source code and documentation, is in the
61  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
```

```

62 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
63 * for further information.
64 * </blockquote>
65 *
66 * <p>This class wraps a SAX1 {@link org.xml.sax.Parser Parser}
67 * and makes it act as a SAX2 {@link org.xml.sax.XMLReader XMLReader},
68 * with feature, property, and Namespace support. Note
69 * that it is not possible to report {@link org.xml.sax.ContentHandler#skippedEntity
70 * skippedEntity} events, since SAX1 does not make that information available.</p>
71 *
72 * <p>This adapter does not test for duplicate Namespace-qualified
73 * attribute names.</p>
74 *
75 * @since SAX 2.0
76 * @author David Megginson
77 * @version 2.0.1 (sax2r2)
78 * @see org.xml.sax.helpers.XMLReaderAdapter
79 * @see org.xml.sax.XMLReader
80 * @see org.xml.sax.Parser
81 */
82 public class ParserAdapter implements XMLReader, DocumentHandler
83 {
84     private static SecuritySupport ss = new SecuritySupport();
85
86     ///////////////////////////////////////////////////
87     // Constructors.
88     ///////////////////////////////////////////////////
89
90
91     /**
92     * Construct a new parser adapter.
93     *
94     * <p>Use the "org.xml.sax.parser" property to locate the
95     * embedded SAX1 driver.</p>
96     *
97     * @exception SAXException If the embedded driver
98     *         cannot be instantiated or if the
99     *         org.xml.sax.parser property is not specified.
100    */
101    public ParserAdapter ()
102        throws SAXException
103    {
104        super();
105
106        String driver = ss.getProperty("org.xml.sax.parser");
107
108        try {
109            setup(ParserFactory.makeParser());
110        } catch (ClassNotFoundException e1) {
111            throw new
112                SAXException("Cannot find SAX1 driver class " +
113                    driver, e1);
114        } catch (IllegalAccessException e2) {

```

```

115         throw new
116             SAXException("SAX1 driver class " +
117                 driver +
118                 " found but cannot be loaded", e2);
119     } catch (InstantiationException e3) {
120         throw new
121             SAXException("SAX1 driver class " +
122                 driver +
123                 " loaded but cannot be instantiated", e3);
124     } catch (ClassCastException e4) {
125         throw new
126             SAXException("SAX1 driver class " +
127                 driver +
128                 " does not implement org.xml.sax.Parser");
129     } catch (NullPointerException e5) {
130         throw new
131             SAXException("System property org.xml.sax.parser not specified");
132     }
133 }
134
135
136 /**
137  * Construct a new parser adapter.
138  *
139  * <p>Note that the embedded parser cannot be changed once the
140  * adapter is created; to embed a different parser, allocate
141  * a new ParserAdapter.</p>
142  *
143  * @param parser The SAX1 parser to embed.
144  * @exception java.lang.NullPointerException If the parser parameter
145  *         is null.
146  */
147 public ParserAdapter (Parser parser)
148 {
149     super();
150     setup(parser);
151 }
152
153
154 /**
155  * Internal setup method.
156  *
157  * @param parser The embedded parser.
158  * @exception java.lang.NullPointerException If the parser parameter
159  *         is null.
160  */
161 private void setup (Parser parser)
162 {
163     if (parser == null) {
164         throw new
165             NullPointerException("Parser argument must not be null");
166     }
167     this.parser = parser;

```

```

168     atts = new AttributesImpl();
169     nsSupport = new NamespaceSupport();
170     attAdapter = new AttributeListAdapter();
171 }
172
173
174
175 ///////////////////////////////////////////////////////////////////
176 // Implementation of org.xml.sax.XMLReader.
177 ///////////////////////////////////////////////////////////////////
178
179
180 //
181 // Internal constants for the sake of convenience.
182 //
183 private final static String FEATURES = "http://xml.org/sax/features/";
184 private final static String NAMESPACES = FEATURES + "namespaces";
185 private final static String NAMESPACE_PREFIXES = FEATURES + "namespace-prefixes";
186 private final static String XMLNS_URIs = FEATURES + "xmlns-uris";
187
188
189 /**
190  * Set a feature flag for the parser.
191  *
192  * <p>The only features recognized are namespaces and
193  * namespace-prefixes.</p>
194  *
195  * @param name The feature name, as a complete URI.
196  * @param value The requested feature value.
197  * @exception SAXNotRecognizedException If the feature
198  *         can't be assigned or retrieved.
199  * @exception SAXNotSupportedException If the feature
200  *         can't be assigned that value.
201  * @see org.xml.sax.XMLReader#setFeature
202  */
203 public void setFeature (String name, boolean value)
204     throws SAXNotRecognizedException, SAXNotSupportedException
205 {
206     if (name.equals(NAMESPACES)) {
207         checkNotParsing("feature", name);
208         namespaces = value;
209         if (!namespaces && !prefixes) {
210             prefixes = true;
211         }
212     } else if (name.equals(NAMESPACE_PREFIXES)) {
213         checkNotParsing("feature", name);
214         prefixes = value;
215         if (!prefixes && !namespaces) {
216             namespaces = true;
217         }
218     } else if (name.equals(XMLNS_URIs)) {
219         checkNotParsing("feature", name);
220         uris = value;

```

```
221     } else {
222         throw new SAXNotRecognizedException("Feature: " + name);
223     }
224 }
225
226
227 /**
228  * Check a parser feature flag.
229  *
230  * <p>The only features recognized are namespaces and
231  * namespace-prefixes.</p>
232  *
233  * @param name The feature name, as a complete URI.
234  * @return The current feature value.
235  * @exception SAXNotRecognizedException If the feature
236  *         value can't be assigned or retrieved.
237  * @exception SAXNotSupportedException If the
238  *         feature is not currently readable.
239  * @see org.xml.sax.XMLReader#setFeature
240  */
241 public boolean getFeature (String name)
242     throws SAXNotRecognizedException, SAXNotSupportedException
243 {
244     if (name.equals(NAMESPACES)) {
245         return namespaces;
246     } else if (name.equals(NAMESPACE_PREFIXES)) {
247         return prefixes;
248     } else if (name.equals(XMLNS_URIs)) {
249         return uris;
250     } else {
251         throw new SAXNotRecognizedException("Feature: " + name);
252     }
253 }
254
255
256 /**
257  * Set a parser property.
258  *
259  * <p>No properties are currently recognized.</p>
260  *
261  * @param name The property name.
262  * @param value The property value.
263  * @exception SAXNotRecognizedException If the property
264  *         value can't be assigned or retrieved.
265  * @exception SAXNotSupportedException If the property
266  *         can't be assigned that value.
267  * @see org.xml.sax.XMLReader#setProperty
268  */
269 public void setProperty (String name, Object value)
270     throws SAXNotRecognizedException, SAXNotSupportedException
271 {
272     throw new SAXNotRecognizedException("Property: " + name);
273 }
```

```
274
275
276 /**
277  * Get a parser property.
278  *
279  * <p>No properties are currently recognized.</p>
280  *
281  * @param name The property name.
282  * @return The property value.
283  * @exception SAXNotRecognizedException If the property
284  *         value can't be assigned or retrieved.
285  * @exception SAXNotSupportedException If the property
286  *         value is not currently readable.
287  * @see org.xml.sax.XMLReader#getProperty
288  */
289 public Object getProperty (String name)
290     throws SAXNotRecognizedException, SAXNotSupportedException
291 {
292     throw new SAXNotRecognizedException("Property: " + name);
293 }
294
295
296 /**
297  * Set the entity resolver.
298  *
299  * @param resolver The new entity resolver.
300  * @see org.xml.sax.XMLReader#setEntityResolver
301  */
302 public void setEntityResolver (EntityResolver resolver)
303 {
304     entityResolver = resolver;
305 }
306
307
308 /**
309  * Return the current entity resolver.
310  *
311  * @return The current entity resolver, or null if none was supplied.
312  * @see org.xml.sax.XMLReader#getEntityResolver
313  */
314 public EntityResolver getEntityResolver ()
315 {
316     return entityResolver;
317 }
318
319
320 /**
321  * Set the DTD handler.
322  *
323  * @param handler the new DTD handler
324  * @see org.xml.sax.XMLReader#setEntityResolver
325  */
326 public void setDTDHandler (DTDHandler handler)
```



```
327     {
328         dtdHandler = handler;
329     }
330
331
332     /**
333      * Return the current DTD handler.
334      *
335      * @return the current DTD handler, or null if none was supplied
336      * @see org.xml.sax.XMLReader#getEntityResolver
337      */
338     public DTDHandler getDTDHandler ()
339     {
340         return dtdHandler;
341     }
342
343
344     /**
345      * Set the content handler.
346      *
347      * @param handler the new content handler
348      * @see org.xml.sax.XMLReader#setEntityResolver
349      */
350     public void setContentHandler (ContentHandler handler)
351     {
352         contentHandler = handler;
353     }
354
355
356     /**
357      * Return the current content handler.
358      *
359      * @return The current content handler, or null if none was supplied.
360      * @see org.xml.sax.XMLReader#getEntityResolver
361      */
362     public ContentHandler getContentHandler ()
363     {
364         return contentHandler;
365     }
366
367
368     /**
369      * Set the error handler.
370      *
371      * @param handler The new error handler.
372      * @see org.xml.sax.XMLReader#setEntityResolver
373      */
374     public void setErrorHandler (ErrorHandler handler)
375     {
376         errorHandler = handler;
377     }
378
379
```

```
380 /**
381  * Return the current error handler.
382  *
383  * @return The current error handler, or null if none was supplied.
384  * @see org.xml.sax.XMLReader#getEntityResolver
385  */
386 public ErrorHandler getErrorHandler ()
387 {
388     return errorHandler;
389 }
390
391
392 /**
393  * Parse an XML document.
394  *
395  * @param systemId The absolute URL of the document.
396  * @exception java.io.IOException If there is a problem reading
397  *         the raw content of the document.
398  * @exception SAXException If there is a problem
399  *         processing the document.
400  * @see #parse(org.xml.sax.InputSource)
401  * @see org.xml.sax.Parser#parse(java.lang.String)
402  */
403 public void parse (String systemId)
404     throws IOException, SAXException
405 {
406     parse(new InputSource(systemId));
407 }
408
409
410 /**
411  * Parse an XML document.
412  *
413  * @param input An input source for the document.
414  * @exception java.io.IOException If there is a problem reading
415  *         the raw content of the document.
416  * @exception SAXException If there is a problem
417  *         processing the document.
418  * @see #parse(java.lang.String)
419  * @see org.xml.sax.Parser#parse(org.xml.sax.InputSource)
420  */
421 public void parse (InputSource input)
422     throws IOException, SAXException
423 {
424     if (parsing) {
425         throw new SAXException("Parser is already in use");
426     }
427     setupParser();
428     parsing = true;
429     try {
430         parser.parse(input);
431     } finally {
432         parsing = false;
```

```

433     }
434     parsing = false;
435 }
436
437
438
439 ////////////////////////////////////////////////////
440 // Implementation of org.xml.sax.DocumentHandler.
441 ////////////////////////////////////////////////////
442
443
444 /**
445  * Adapter implementation method; do not call.
446  * Adapt a SAX1 document locator event.
447  *
448  * @param locator A document locator.
449  * @see org.xml.sax.ContentHandler#setDocumentLocator
450  */
451 public void setDocumentLocator (Locator locator)
452 {
453     this.locator = locator;
454     if (contentHandler != null) {
455         contentHandler.setDocumentLocator(locator);
456     }
457 }
458
459
460 /**
461  * Adapter implementation method; do not call.
462  * Adapt a SAX1 start document event.
463  *
464  * @exception SAXException The client may raise a
465  *         processing exception.
466  * @see org.xml.sax.DocumentHandler#startDocument
467  */
468 public void startDocument ()
469     throws SAXException
470 {
471     if (contentHandler != null) {
472         contentHandler.startDocument();
473     }
474 }
475
476
477 /**
478  * Adapter implementation method; do not call.
479  * Adapt a SAX1 end document event.
480  *
481  * @exception SAXException The client may raise a
482  *         processing exception.
483  * @see org.xml.sax.DocumentHandler#endDocument
484  */
485 public void endDocument ()

```

```

486     throws SAXException
487 {
488     if (contentHandler != null) {
489         contentHandler.endDocument();
490     }
491 }
492
493
494 /**
495  * Adapter implementation method; do not call.
496  * Adapt a SAX1 startElement event.
497  *
498  * <p>If necessary, perform Namespace processing.</p>
499  *
500  * @param qName The qualified (prefixed) name.
501  * @param qAtts The XML attribute list (with qnames).
502  * @exception SAXException The client may raise a
503  *         processing exception.
504  */
505 public void startElement (String qName, AttributeList qAtts)
506     throws SAXException
507 {
508     // These are exceptions from the
509     // first pass; they should be
510     // ignored if there's a second pass,
511     // but reported otherwise.
512     Vector exceptions = null;
513
514     // If we're not doing Namespace
515     // processing, dispatch this quickly.
516     if (!namespaces) {
517         if (contentHandler != null) {
518             attAdapter.setAttributeList(qAtts);
519             contentHandler.startElement("", "", qName.intern(),
520                                     attAdapter);
521         }
522         return;
523     }
524
525     // OK, we're doing Namespace processing.
526     nsSupport.pushContext();
527     int length = qAtts.getLength();
528
529     // First pass: handle NS decls
530     for (int i = 0; i < length; i++) {
531         String attQName = qAtts.getName(i);
532
533         if (!attQName.startsWith("xmlns"))
534             continue;
535
536         // Could be a declaration...
537         String prefix;
538         int n = attQName.indexOf(':');

```

```

539
540             // xmlns=...
541     if (n == -1 && attQName.length () == 5) {
542         prefix = "";
543     } else if (n != 5) {
544         // XML namespaces spec doesn't discuss "xmlnsf:oo"
545         // (and similarly named) attributes ... at most, warn
546         continue;
547     } else // xmlns:foo=...
548         prefix = attQName.substring(n+1);
549
550     String value = qAtts.getValue(i);
551     if (!nsSupport.declarePrefix(prefix, value)) {
552         reportError("Illegal Namespace prefix: " + prefix);
553         continue;
554     }
555     if (contentHandler != null)
556         contentHandler.startPrefixMapping(prefix, value);
557 }
558
559             // Second pass: copy all relevant
560             // attributes into the SAX2 AttributeList
561             // using updated prefix bindings
562     atts.clear();
563     for (int i = 0; i < length; i++) {
564         String attQName = qAtts.getName(i);
565         String type = qAtts.getType(i);
566         String value = qAtts.getValue(i);
567
568             // Declaration?
569     if (attQName.startsWith("xmlns")) {
570         String prefix;
571         int n = attQName.indexOf(':');
572
573         if (n == -1 && attQName.length () == 5) {
574             prefix = "";
575         } else if (n != 5) {
576             // XML namespaces spec doesn't discuss "xmlnsf:oo"
577             // (and similarly named) attributes ... ignore
578             prefix = null;
579         } else {
580             prefix = attQName.substring(6);
581         }
582
583             // Yes, decl: report or prune
584     if (prefix != null) {
585         if (prefixes) {
586             if (uris)
587                 // note funky case: localname can be null
588                 // when declaring the default prefix, and
589                 // yet the uri isn't null.
590                 atts.addAttribute (nsSupport.XMLNS, prefix,
591                                     attQName.intern(), type, value);
592             else

```

```

592         atts.addAttribute("", "",
593             attQName.intern(), type, value);
594     }
595     continue;
596 }
597 }
598
599     // Not a declaration -- report
600     try {
601         String attName[] = processName(attQName, true, true);
602         atts.addAttribute(attName[0], attName[1], attName[2],
603             type, value);
604     } catch (SAXException e) {
605         if (exceptions == null)
606             exceptions = new Vector();
607         exceptions.addElement(e);
608         atts.addAttribute("", attQName, attQName, type, value);
609     }
610 }
611
612 // now handle the deferred exception reports
613 if (exceptions != null && errorHandler != null) {
614     for (int i = 0; i < exceptions.size(); i++)
615         errorHandler.error((SAXParseException)
616             exceptions.elementAt(i));
617 }
618
619 // OK, finally report the event.
620 if (contentHandler != null) {
621     String name[] = processName(qName, false, false);
622     contentHandler.startElement(name[0], name[1], name[2], atts);
623 }
624 }
625
626
627 /**
628  * Adapter implementation method; do not call.
629  * Adapt a SAX1 end element event.
630  *
631  * @param qName The qualified (prefixed) name.
632  * @exception SAXException The client may raise a
633  *         processing exception.
634  * @see org.xml.sax.DocumentHandler#endElement
635  */
636 public void endElement (String qName)
637     throws SAXException
638 {
639     // If we're not doing Namespace
640     // processing, dispatch this quickly.
641     if (!namespaces) {
642         if (contentHandler != null) {
643             contentHandler.endElement("", "", qName.intern());
644         }

```

```
645         return;
646     }
647
648         // Split the name.
649     String names[] = processName(qName, false, false);
650     if (contentHandler != null) {
651         contentHandler.endElement(names[0], names[1], names[2]);
652         Enumeration prefixes = nsSupport.getDeclaredPrefixes();
653         while (prefixes.hasMoreElements()) {
654             String prefix = (String)prefixes.nextElement();
655             contentHandler.endPrefixMapping(prefix);
656         }
657     }
658     nsSupport.popContext();
659 }
660
661 /**
662  * Adapter implementation method; do not call.
663  * Adapt a SAX1 characters event.
664  *
665  * @param ch An array of characters.
666  * @param start The starting position in the array.
667  * @param length The number of characters to use.
668  * @exception SAXException The client may raise a
669  *             processing exception.
670  * @see org.xml.sax.DocumentHandler#characters
671  */
672 public void characters (char ch[], int start, int length)
673     throws SAXException
674 {
675     if (contentHandler != null) {
676         contentHandler.characters(ch, start, length);
677     }
678 }
679
680 /**
681  * Adapter implementation method; do not call.
682  * Adapt a SAX1 ignorable whitespace event.
683  *
684  * @param ch An array of characters.
685  * @param start The starting position in the array.
686  * @param length The number of characters to use.
687  * @exception SAXException The client may raise a
688  *             processing exception.
689  * @see org.xml.sax.DocumentHandler#ignorableWhitespace
690  */
691 public void ignorableWhitespace (char ch[], int start, int length)
692     throws SAXException
693 {
694     if (contentHandler != null) {
695         contentHandler.ignorableWhitespace(ch, start, length);
696     }
697 }
```

```
698     }
699 }
700
701
702 /**
703  * Adapter implementation method; do not call.
704  * Adapt a SAX1 processing instruction event.
705  *
706  * @param target The processing instruction target.
707  * @param data The remainder of the processing instruction
708  * @exception SAXException The client may raise a
709  *             processing exception.
710  * @see org.xml.sax.DocumentHandler#processingInstruction
711  */
712 public void processingInstruction (String target, String data)
713     throws SAXException
714 {
715     if (contentHandler != null) {
716         contentHandler.processingInstruction(target, data);
717     }
718 }
719
720
721
722 ///////////////////////////////////////////////////
723 // Internal utility methods.
724 ///////////////////////////////////////////////////
725
726
727 /**
728  * Initialize the parser before each run.
729  */
730 private void setupParser ()
731 {
732     // catch an illegal "nonsense" state.
733     if (!prefixes && !namespaces)
734         throw new IllegalStateException ();
735
736     nsSupport.reset();
737     if (uris)
738         nsSupport.setNamespaceDeclUris (true);
739
740     if (entityResolver != null) {
741         parser.setEntityResolver(entityResolver);
742     }
743     if (dtdHandler != null) {
744         parser.setDTDHandler(dtdHandler);
745     }
746     if (errorHandler != null) {
747         parser.setErrorHandler(errorHandler);
748     }
749     parser.setDocumentHandler(this);
750     locator = null;
```



```

751     }
752
753
754     /**
755      * Process a qualified (prefixed) name.
756      *
757      * <p>If the name has an undeclared prefix, use only the qname
758      * and make an ErrorHandler.error callback in case the app is
759      * interested.</p>
760      *
761      * @param qName The qualified (prefixed) name.
762      * @param isAttribute true if this is an attribute name.
763      * @return The name split into three parts.
764      * @exception SAXException The client may throw
765      *         an exception if there is an error callback.
766      */
767     private String [] processName (String qName, boolean isAttribute,
768                                   boolean useException)
769         throws SAXException
770     {
771         String parts[] = nsSupport.processName(qName, nameParts,
772                                                isAttribute);
773         if (parts == null) {
774             if (useException)
775                 throw makeException("Undeclared prefix: " + qName);
776             reportError("Undeclared prefix: " + qName);
777             parts = new String[3];
778             parts[0] = parts[1] = "";
779             parts[2] = qName.intern();
780         }
781         return parts;
782     }
783
784
785     /**
786      * Report a non-fatal error.
787      *
788      * @param message The error message.
789      * @exception SAXException The client may throw
790      *         an exception.
791      */
792     void reportError (String message)
793         throws SAXException
794     {
795         if (errorHandler != null)
796             errorHandler.error(makeException(message));
797     }
798
799
800     /**
801      * Construct an exception for the current context.
802      *
803      * @param message The error message.

```

```

804     */
805     private SAXParseException makeException (String message)
806     {
807         if (locator != null) {
808             return new SAXParseException(message, locator);
809         } else {
810             return new SAXParseException(message, null, null, -1, -1);
811         }
812     }
813
814
815     /**
816      * Throw an exception if we are parsing.
817      *
818      * <p>Use this method to detect illegal feature or
819      * property changes.</p>
820      *
821      * @param type The type of thing (feature or property).
822      * @param name The feature or property name.
823      * @exception SAXNotSupportedException If a
824      *         document is currently being parsed.
825      */
826     private void checkNotParsing (String type, String name)
827         throws SAXNotSupportedException
828     {
829         if (parsing) {
830             throw new SAXNotSupportedException("Cannot change " +
831                 type + ' ' +
832                 name + " while parsing");
833         }
834     }
835 }
836
837
838
839     //////////////////////////////////////
840     // Internal state.
841     //////////////////////////////////////
842
843     private NamespaceSupport nsSupport;
844     private AttributeListAdapter attAdapter;
845
846     private boolean parsing = false;
847     private String nameParts[] = new String[3];
848
849     private Parser parser = null;
850
851     private AttributesImpl atts = null;
852
853         // Features
854     private boolean namespaces = true;
855     private boolean prefixes = false;
856     private boolean uris = false;

```

```

857
858             // Properties
859
860             // Handlers
861 Locator locator;
862
863 EntityResolver entityResolver = null;
864 DTDHandler dtdHandler = null;
865 ContentHandler contentHandler = null;
866 ErrorHandler errorHandler = null;
867
868
869
870 ////////////////////////////////////////////////////
871 // Inner class to wrap an AttributeList when not doing NS proc.
872 ////////////////////////////////////////////////////
873
874
875 /**
876  * Adapt a SAX1 AttributeList as a SAX2 Attributes object.
877  *
878  * <p>This class is in the Public Domain, and comes with NO
879  * WARRANTY of any kind.</p>
880  *
881  * <p>This wrapper class is used only when Namespace support
882  * is disabled -- it provides pretty much a direct mapping
883  * from SAX1 to SAX2, except that names and types are
884  * interned whenever requested.</p>
885  */
886 final class AttributeListAdapter implements Attributes
887 {
888
889     /**
890      * Construct a new adapter.
891      */
892     AttributeListAdapter ()
893     {
894
895
896
897     /**
898      * Set the embedded AttributeList.
899      *
900      * <p>This method must be invoked before any of the others
901      * can be used.</p>
902      *
903      * @param The SAX1 attribute list (with qnames).
904      */
905     void setAttributeList (AttributeList qAtts)
906     {
907         this.qAtts = qAtts;
908     }
909

```

```
910
911     /**
912      * Return the length of the attribute list.
913      *
914      * @return The number of attributes in the list.
915      * @see org.xml.sax.Attributes#getLength
916      */
917     public int getLength ()
918     {
919         return qAtts.getLength();
920     }
921
922
923     /**
924      * Return the Namespace URI of the specified attribute.
925      *
926      * @param The attribute's index.
927      * @return Always the empty string.
928      * @see org.xml.sax.Attributes#getURI
929      */
930     public String getURI (int i)
931     {
932         return "";
933     }
934
935
936     /**
937      * Return the local name of the specified attribute.
938      *
939      * @param The attribute's index.
940      * @return Always the empty string.
941      * @see org.xml.sax.Attributes#getLocalName
942      */
943     public String getLocalName (int i)
944     {
945         return "";
946     }
947
948
949     /**
950      * Return the qualified (prefixed) name of the specified attribute.
951      *
952      * @param The attribute's index.
953      * @return The attribute's qualified name, internalized.
954      */
955     public String getQName (int i)
956     {
957         return qAtts.getName(i).intern();
958     }
959
960
961     /**
962      * Return the type of the specified attribute.
```

```
963      *
964      * @param The attribute's index.
965      * @return The attribute's type as an internalized string.
966      */
967     public String getType (int i)
968     {
969         return qAtts.getType(i).intern();
970     }
971
972
973     /**
974      * Return the value of the specified attribute.
975      *
976      * @param The attribute's index.
977      * @return The attribute's value.
978      */
979     public String getValue (int i)
980     {
981         return qAtts.getValue(i);
982     }
983
984
985     /**
986      * Look up an attribute index by Namespace name.
987      *
988      * @param uri The Namespace URI or the empty string.
989      * @param localName The local name.
990      * @return The attributes index, or -1 if none was found.
991      * @see org.xml.sax.Attributes#getIndex(java.lang.String,java.lang.String)
992      */
993     public int getIndex (String uri, String localName)
994     {
995         return -1;
996     }
997
998
999     /**
1000      * Look up an attribute index by qualified (prefixed) name.
1001      *
1002      * @param qName The qualified name.
1003      * @return The attributes index, or -1 if none was found.
1004      * @see org.xml.sax.Attributes#getIndex(java.lang.String)
1005      */
1006     public int getIndex (String qName)
1007     {
1008         int max = atts.getLength();
1009         for (int i = 0; i < max; i++) {
1010             if (qAtts.getName(i).equals(qName)) {
1011                 return i;
1012             }
1013         }
1014         return -1;
1015     }
```

```
1016
1017
1018     /**
1019      * Look up the type of an attribute by Namespace name.
1020      *
1021      * @param uri The Namespace URI
1022      * @param localName The local name.
1023      * @return The attribute's type as an internalized string.
1024      */
1025     public String getType (String uri, String localName)
1026     {
1027         return null;
1028     }
1029
1030
1031     /**
1032      * Look up the type of an attribute by qualified (prefixed) name.
1033      *
1034      * @param qName The qualified name.
1035      * @return The attribute's type as an internalized string.
1036      */
1037     public String getType (String qName)
1038     {
1039         return qAtts.getType(qName).intern();
1040     }
1041
1042
1043     /**
1044      * Look up the value of an attribute by Namespace name.
1045      *
1046      * @param uri The Namespace URI
1047      * @param localName The local name.
1048      * @return The attribute's value.
1049      */
1050     public String getValue (String uri, String localName)
1051     {
1052         return null;
1053     }
1054
1055
1056     /**
1057      * Look up the value of an attribute by qualified (prefixed) name.
1058      *
1059      * @param qName The qualified name.
1060      * @return The attribute's value.
1061      */
1062     public String getValue (String qName)
1063     {
1064         return qAtts.getValue(qName);
1065     }
1066
1067     private AttributeList qAtts;
1068 }
```

```
1069 }
1070
1071 // end of ParserAdapter.java
```

## 49.8 ParserFactory.java

Secuencia 510: org/xml/sax/helpers/ParserFactory.java

```
1  /*
2  * Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // SAX parser factory.
27 // http://www.saxproject.org
28 // No warranty; no copyright -- use this as you will.
29 // $Id: ParserFactory.java,v 1.2 2004/11/03 22:53:09 jsuttor Exp $
30
31 package org.xml.sax.helpers;
32
33 import org.xml.sax.Parser;
34
35
36 /**
37 * Java-specific class for dynamically loading SAX parsers.
38 *
39 * <blockquote>
40 * <em>This module, both source code and documentation, is in the
41 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
42 * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
43 * for further information.
44 * </blockquote>
45 *
46 * <p><strong>Note:</strong> This class is designed to work with the now-deprecated
```

```

47 * SAX1 {@link org.xml.sax.Parser Parser} class. SAX2 applications should use
48 * {@link org.xml.sax.helpers.XMLReaderFactory XMLReaderFactory} instead.</p>
49 *
50 * <p>ParserFactory is not part of the platform-independent definition
51 * of SAX; it is an additional convenience class designed
52 * specifically for Java XML application writers. SAX applications
53 * can use the static methods in this class to allocate a SAX parser
54 * dynamically at run-time based either on the value of the
55 * `org.xml.sax.parser' system property or on a string containing the class
56 * name.</p>
57 *
58 * <p>Note that the application still requires an XML parser that
59 * implements SAX1.</p>
60 *
61 * @deprecated This class works with the deprecated
62 *             {@link org.xml.sax.Parser Parser}
63 *             interface.
64 * @since SAX 1.0
65 * @author David Megginson
66 * @version 2.0.1 (sax2r2)
67 */
68 public class ParserFactory {
69     private static SecuritySupport ss = new SecuritySupport();
70
71     /**
72      * Private null constructor.
73      */
74     private ParserFactory ()
75     {
76     }
77
78
79     /**
80      * Create a new SAX parser using the `org.xml.sax.parser' system property.
81      *
82      * <p>The named class must exist and must implement the
83      * {@link org.xml.sax.Parser Parser} interface.</p>
84      *
85      * @exception java.lang.NullPointerException There is no value
86      *             for the `org.xml.sax.parser' system property.
87      * @exception java.lang.ClassNotFoundException The SAX parser
88      *             class was not found (check your CLASSPATH).
89      * @exception IllegalAccessException The SAX parser class was
90      *             found, but you do not have permission to load
91      *             it.
92      * @exception InstantiationException The SAX parser class was
93      *             found but could not be instantiated.
94      * @exception java.lang.ClassCastException The SAX parser class
95      *             was found and instantiated, but does not implement
96      *             org.xml.sax.Parser.
97      * @see #makeParser(java.lang.String)
98      * @see org.xml.sax.Parser
99      */

```



```

100     public static Parser makeParser ()
101         throws ClassNotFoundException,
102             IllegalAccessException,
103             InstantiationException,
104             NullPointerException,
105             ClassCastException
106     {
107         String className = ss.getProperty("org.xml.sax.parser");
108         if (className == null) {
109             throw new NullPointerException("No value for sax.parser property");
110         } else {
111             return makeParser(className);
112         }
113     }
114
115
116     /**
117     * Create a new SAX parser object using the class name provided.
118     *
119     * <p>The named class must exist and must implement the
120     * {@link org.xml.sax.Parser Parser} interface.</p>
121     *
122     * @param className A string containing the name of the
123     *                 SAX parser class.
124     * @exception java.lang.ClassNotFoundException The SAX parser
125     *                 class was not found (check your CLASSPATH).
126     * @exception IllegalAccessException The SAX parser class was
127     *                 found, but you do not have permission to load
128     *                 it.
129     * @exception InstantiationException The SAX parser class was
130     *                 found but could not be instantiated.
131     * @exception java.lang.ClassCastException The SAX parser class
132     *                 was found and instantiated, but does not implement
133     *                 org.xml.sax.Parser.
134     * @see #makeParser()
135     * @see org.xml.sax.Parser
136     */
137     public static Parser makeParser (String className)
138         throws ClassNotFoundException,
139             IllegalAccessException,
140             InstantiationException,
141             ClassCastException
142     {
143         return (Parser) NewInstance.newInstance (
144             ss.getContextClassLoader(), className);
145     }
146
147 }

```

## 49.9 SecuritySupport.java

Secuencia 511: org/xml/sax/helpers/SecuritySupport.java

1 /\*

```
2  * Copyright (c) 2004, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 package org.xml.sax.helpers;
27
28 import java.io.*;
29 import java.security.*;
30
31 /**
32  * This class is duplicated for each JAXP subpackage so keep it in sync.
33  * It is package private and therefore is not exposed as part of the JAXP
34  * API.
35  *
36  * Security related methods that only work on J2SE 1.2 and newer.
37  */
38 class SecuritySupport {
39
40
41     ClassLoader getContextClassLoader() throws SecurityException{
42         return (ClassLoader)
43             AccessController.doPrivileged(new PrivilegedAction() {
44                 public Object run() {
45                     ClassLoader cl = null;
46                     //try {
47                         cl = Thread.currentThread().getContextClassLoader();
48                     //} catch (SecurityException ex) { }
49
50                     if (cl == null)
51                         cl = ClassLoader.getSystemClassLoader();
52
53                     return cl;
54                 }
55             })
56     }
```

```
55     });
56 }
57
58 String getSystemProperty(final String propName) {
59     return (String)
60         AccessController.doPrivileged(new PrivilegedAction() {
61             public Object run() {
62                 return System.getProperty(propName);
63             }
64         });
65 }
66
67 FileInputStream getFileInputStream(final File file)
68     throws FileNotFoundException
69 {
70     try {
71         return (FileInputStream)
72             AccessController.doPrivileged(new PrivilegedExceptionAction() {
73                 public Object run() throws FileNotFoundException {
74                     return new FileInputStream(file);
75                 }
76             });
77     } catch (PrivilegedActionException e) {
78         throw (FileNotFoundException)e.getException();
79     }
80 }
81
82 InputStream getResourceAsStream(final ClassLoader cl,
83                                 final String name)
84 {
85     return (InputStream)
86         AccessController.doPrivileged(new PrivilegedAction() {
87             public Object run() {
88                 InputStream ris;
89                 if (cl == null) {
90                     ris = Object.class.getResourceAsStream(name);
91                 } else {
92                     ris = cl.getResourceAsStream(name);
93                 }
94                 return ris;
95             }
96         });
97 }
98
99 boolean doesFileExist(final File f) {
100     return ((Boolean)
101         AccessController.doPrivileged(new PrivilegedAction() {
102             public Object run() {
103                 return new Boolean(f.exists());
104             }
105         })).booleanValue();
106 }
107
```

108 }

## 49.10 XMLFilterImpl.java

Secuencia 512: org/xml/sax/helpers/XMLFilterImpl.java

```
1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // XMLFilterImpl.java - base SAX2 filter implementation.
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the Public Domain.
30 // $Id: XMLFilterImpl.java,v 1.3 2004/11/03 22:53:09 jsuttor Exp $
31
32 package org.xml.sax.helpers;
33
34 import java.io.IOException;
35
36 import org.xml.sax.XMLReader;
37 import org.xml.sax.XMLFilter;
38 import org.xml.sax.InputSource;
39 import org.xml.sax.Locator;
40 import org.xml.sax.Attributes;
41 import org.xml.sax.EntityResolver;
42 import org.xml.sax.DTDHandler;
43 import org.xml.sax.ContentHandler;
44 import org.xml.sax.ErrorHandler;
45 import org.xml.sax.SAXException;
46 import org.xml.sax.SAXParseException;
47 import org.xml.sax.SAXNotSupportedException;
48 import org.xml.sax.SAXNotRecognizedException;
```

```

49
50
51 /**
52  * Base class for deriving an XML filter.
53  *
54  * <blockquote>
55  * <em>This module, both source code and documentation, is in the
56  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
57  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
58  * for further information.
59  * </blockquote>
60  *
61  * <p>This class is designed to sit between an {@link org.xml.sax.XMLReader
62  * XMLReader} and the client application's event handlers. By default, it
63  * does nothing but pass requests up to the reader and events
64  * on to the handlers unmodified, but subclasses can override
65  * specific methods to modify the event stream or the configuration
66  * requests as they pass through.</p>
67  *
68  * @since SAX 2.0
69  * @author David Megginson
70  * @see org.xml.sax.XMLFilter
71  * @see org.xml.sax.XMLReader
72  * @see org.xml.sax.EntityResolver
73  * @see org.xml.sax.DTDHandler
74  * @see org.xml.sax.ContentHandler
75  * @see org.xml.sax.ErrorHandler
76  */
77 public class XMLFilterImpl
78     implements XMLFilter, EntityResolver, DTDHandler, ContentHandler, ErrorHandler
79 {
80
81
82     //////////////////////////////////////
83     // Constructors.
84     //////////////////////////////////////
85
86
87     /**
88      * Construct an empty XML filter, with no parent.
89      *
90      * <p>This filter will have no parent: you must assign a parent
91      * before you start a parse or do any configuration with
92      * setFeature or setProperty, unless you use this as a pure event
93      * consumer rather than as an {@link XMLReader}.</p>
94      *
95      * @see org.xml.sax.XMLReader#setFeature
96      * @see org.xml.sax.XMLReader#setProperty
97      * @see #setParent
98      */
99     public XMLFilterImpl ()
100     {
101         super();

```

```
102     }
103
104
105     /**
106      * Construct an XML filter with the specified parent.
107      *
108      * @see #setParent
109      * @see #getParent
110      */
111     public XMLFilterImpl (XMLReader parent)
112     {
113         super();
114         setParent(parent);
115     }
116
117
118
119     //////////////////////////////////////
120     // Implementation of org.xml.sax.XMLFilter.
121     //////////////////////////////////////
122
123
124     /**
125      * Set the parent reader.
126      *
127      * <p>This is the {@link org.xml.sax.XMLReader XMLReader} from which
128      * this filter will obtain its events and to which it will pass its
129      * configuration requests. The parent may itself be another filter.</p>
130      *
131      * <p>If there is no parent reader set, any attempt to parse
132      * or to set or get a feature or property will fail.</p>
133      *
134      * @param parent The parent XML reader.
135      * @see #getParent
136      */
137     public void setParent (XMLReader parent)
138     {
139         this.parent = parent;
140     }
141
142
143     /**
144      * Get the parent reader.
145      *
146      * @return The parent XML reader, or null if none is set.
147      * @see #setParent
148      */
149     public XMLReader getParent ()
150     {
151         return parent;
152     }
153
154
```

```

155
156 ///////////////////////////////////////////////////
157 // Implementation of org.xml.sax.XMLReader.
158 ///////////////////////////////////////////////////
159
160
161 /**
162  * Set the value of a feature.
163  *
164  * <p>This will always fail if the parent is null.</p>
165  *
166  * @param name The feature name.
167  * @param value The requested feature value.
168  * @exception org.xml.sax.SAXNotRecognizedException If the feature
169  *             value can't be assigned or retrieved from the parent.
170  * @exception org.xml.sax.SAXNotSupportedException When the
171  *             parent recognizes the feature name but
172  *             cannot set the requested value.
173  */
174 public void setFeature (String name, boolean value)
175     throws SAXNotRecognizedException, SAXNotSupportedException
176 {
177     if (parent != null) {
178         parent.setFeature(name, value);
179     } else {
180         throw new SAXNotRecognizedException("Feature: " + name);
181     }
182 }
183
184
185 /**
186  * Look up the value of a feature.
187  *
188  * <p>This will always fail if the parent is null.</p>
189  *
190  * @param name The feature name.
191  * @return The current value of the feature.
192  * @exception org.xml.sax.SAXNotRecognizedException If the feature
193  *             value can't be assigned or retrieved from the parent.
194  * @exception org.xml.sax.SAXNotSupportedException When the
195  *             parent recognizes the feature name but
196  *             cannot determine its value at this time.
197  */
198 public boolean getFeature (String name)
199     throws SAXNotRecognizedException, SAXNotSupportedException
200 {
201     if (parent != null) {
202         return parent.getFeature(name);
203     } else {
204         throw new SAXNotRecognizedException("Feature: " + name);
205     }
206 }
207

```

```
208
209 /**
210  * Set the value of a property.
211  *
212  * <p>This will always fail if the parent is null.</p>
213  *
214  * @param name The property name.
215  * @param value The requested property value.
216  * @exception org.xml.sax.SAXNotRecognizedException If the property
217  *             value can't be assigned or retrieved from the parent.
218  * @exception org.xml.sax.SAXNotSupportedException When the
219  *             parent recognizes the property name but
220  *             cannot set the requested value.
221  */
222 public void setProperty (String name, Object value)
223     throws SAXNotRecognizedException, SAXNotSupportedException
224 {
225     if (parent != null) {
226         parent.setProperty(name, value);
227     } else {
228         throw new SAXNotRecognizedException("Property: " + name);
229     }
230 }
231
232
233 /**
234  * Look up the value of a property.
235  *
236  * @param name The property name.
237  * @return The current value of the property.
238  * @exception org.xml.sax.SAXNotRecognizedException If the property
239  *             value can't be assigned or retrieved from the parent.
240  * @exception org.xml.sax.SAXNotSupportedException When the
241  *             parent recognizes the property name but
242  *             cannot determine its value at this time.
243  */
244 public Object getProperty (String name)
245     throws SAXNotRecognizedException, SAXNotSupportedException
246 {
247     if (parent != null) {
248         return parent.getProperty(name);
249     } else {
250         throw new SAXNotRecognizedException("Property: " + name);
251     }
252 }
253
254
255 /**
256  * Set the entity resolver.
257  *
258  * @param resolver The new entity resolver.
259  */
260 public void setEntityResolver (EntityResolver resolver)
```



```
261     {
262         entityResolver = resolver;
263     }
264
265
266     /**
267      * Get the current entity resolver.
268      *
269      * @return The current entity resolver, or null if none was set.
270      */
271     public EntityResolver getEntityResolver ()
272     {
273         return entityResolver;
274     }
275
276
277     /**
278      * Set the DTD event handler.
279      *
280      * @param handler the new DTD handler
281      */
282     public void setDTDHandler (DTDHandler handler)
283     {
284         dtdHandler = handler;
285     }
286
287
288     /**
289      * Get the current DTD event handler.
290      *
291      * @return The current DTD handler, or null if none was set.
292      */
293     public DTDHandler getDTDHandler ()
294     {
295         return dtdHandler;
296     }
297
298
299     /**
300      * Set the content event handler.
301      *
302      * @param handler the new content handler
303      */
304     public void setContentHandler (ContentHandler handler)
305     {
306         contentHandler = handler;
307     }
308
309
310     /**
311      * Get the content event handler.
312      *
313      * @return The current content handler, or null if none was set.
```

```
314     */
315     public ContentHandler getContentHandler ()
316     {
317         return contentHandler;
318     }
319
320
321     /**
322      * Set the error event handler.
323      *
324      * @param handler the new error handler
325      */
326     public void setErrorHandler (ErrorHandler handler)
327     {
328         errorHandler = handler;
329     }
330
331
332     /**
333      * Get the current error event handler.
334      *
335      * @return The current error handler, or null if none was set.
336      */
337     public ErrorHandler getErrorHandler ()
338     {
339         return errorHandler;
340     }
341
342
343     /**
344      * Parse a document.
345      *
346      * @param input The input source for the document entity.
347      * @exception org.xml.sax.SAXException Any SAX exception, possibly
348      *         wrapping another exception.
349      * @exception java.io.IOException An IO exception from the parser,
350      *         possibly from a byte stream or character stream
351      *         supplied by the application.
352      */
353     public void parse (InputSource input)
354         throws SAXException, IOException
355     {
356         setupParse();
357         parent.parse(input);
358     }
359
360
361     /**
362      * Parse a document.
363      *
364      * @param systemId The system identifier as a fully-qualified URI.
365      * @exception org.xml.sax.SAXException Any SAX exception, possibly
366      *         wrapping another exception.
```

```

367     * @exception java.io.IOException An IO exception from the parser,
368     *         possibly from a byte stream or character stream
369     *         supplied by the application.
370     */
371     public void parse (String systemId)
372         throws SAXException, IOException
373     {
374         parse(new InputSource(systemId));
375     }
376
377
378
379     ////////////////////////////////////////////////////
380     // Implementation of org.xml.sax.EntityResolver.
381     ////////////////////////////////////////////////////
382
383
384     /**
385     * Filter an external entity resolution.
386     *
387     * @param publicId The entity's public identifier, or null.
388     * @param systemId The entity's system identifier.
389     * @return A new InputSource or null for the default.
390     * @exception org.xml.sax.SAXException The client may throw
391     *         an exception during processing.
392     * @exception java.io.IOException The client may throw an
393     *         I/O-related exception while obtaining the
394     *         new InputSource.
395     */
396     public InputSource resolveEntity (String publicId, String systemId)
397         throws SAXException, IOException
398     {
399         if (entityResolver != null) {
400             return entityResolver.resolveEntity(publicId, systemId);
401         } else {
402             return null;
403         }
404     }
405
406
407
408     ////////////////////////////////////////////////////
409     // Implementation of org.xml.sax.DTDHandler.
410     ////////////////////////////////////////////////////
411
412
413     /**
414     * Filter a notation declaration event.
415     *
416     * @param name The notation name.
417     * @param publicId The notation's public identifier, or null.
418     * @param systemId The notation's system identifier, or null.
419     * @exception org.xml.sax.SAXException The client may throw

```

```

420     *           an exception during processing.
421     */
422     public void notationDecl (String name, String publicId, String systemId)
423         throws SAXException
424     {
425         if (dtdHandler != null) {
426             dtdHandler.notationDecl(name, publicId, systemId);
427         }
428     }
429
430
431     /**
432     * Filter an unparsed entity declaration event.
433     *
434     * @param name The entity name.
435     * @param publicId The entity's public identifier, or null.
436     * @param systemId The entity's system identifier, or null.
437     * @param notationName The name of the associated notation.
438     * @exception org.xml.sax.SAXException The client may throw
439     *           an exception during processing.
440     */
441     public void unparsedEntityDecl (String name, String publicId,
442                                     String systemId, String notationName)
443         throws SAXException
444     {
445         if (dtdHandler != null) {
446             dtdHandler.unparsedEntityDecl(name, publicId, systemId,
447                                           notationName);
448         }
449     }
450
451
452
453     ////////////////////////////////////////////////////
454     // Implementation of org.xml.sax.ContentHandler.
455     ////////////////////////////////////////////////////
456
457
458     /**
459     * Filter a new document locator event.
460     *
461     * @param locator The document locator.
462     */
463     public void setDocumentLocator (Locator locator)
464     {
465         this.locator = locator;
466         if (contentHandler != null) {
467             contentHandler.setDocumentLocator(locator);
468         }
469     }
470
471
472     /**

```

```
473     * Filter a start document event.
474     *
475     * @exception org.xml.sax.SAXException The client may throw
476     *         an exception during processing.
477     */
478     public void startDocument ()
479         throws SAXException
480     {
481         if (contentHandler != null) {
482             contentHandler.startDocument();
483         }
484     }
485
486
487     /**
488     * Filter an end document event.
489     *
490     * @exception org.xml.sax.SAXException The client may throw
491     *         an exception during processing.
492     */
493     public void endDocument ()
494         throws SAXException
495     {
496         if (contentHandler != null) {
497             contentHandler.endDocument();
498         }
499     }
500
501
502     /**
503     * Filter a start Namespace prefix mapping event.
504     *
505     * @param prefix The Namespace prefix.
506     * @param uri The Namespace URI.
507     * @exception org.xml.sax.SAXException The client may throw
508     *         an exception during processing.
509     */
510     public void startPrefixMapping (String prefix, String uri)
511         throws SAXException
512     {
513         if (contentHandler != null) {
514             contentHandler.startPrefixMapping(prefix, uri);
515         }
516     }
517
518
519     /**
520     * Filter an end Namespace prefix mapping event.
521     *
522     * @param prefix The Namespace prefix.
523     * @exception org.xml.sax.SAXException The client may throw
524     *         an exception during processing.
525     */
```

```
526 public void endPrefixMapping (String prefix)
527     throws SAXException
528 {
529     if (contentHandler != null) {
530         contentHandler.endPrefixMapping(prefix);
531     }
532 }
533
534
535 /**
536  * Filter a start element event.
537  *
538  * @param uri The element's Namespace URI, or the empty string.
539  * @param localName The element's local name, or the empty string.
540  * @param qName The element's qualified (prefixed) name, or the empty
541  *             string.
542  * @param atts The element's attributes.
543  * @exception org.xml.sax.SAXException The client may throw
544  *             an exception during processing.
545  */
546 public void startElement (String uri, String localName, String qName,
547                         Attributes atts)
548     throws SAXException
549 {
550     if (contentHandler != null) {
551         contentHandler.startElement(uri, localName, qName, atts);
552     }
553 }
554
555
556 /**
557  * Filter an end element event.
558  *
559  * @param uri The element's Namespace URI, or the empty string.
560  * @param localName The element's local name, or the empty string.
561  * @param qName The element's qualified (prefixed) name, or the empty
562  *             string.
563  * @exception org.xml.sax.SAXException The client may throw
564  *             an exception during processing.
565  */
566 public void endElement (String uri, String localName, String qName)
567     throws SAXException
568 {
569     if (contentHandler != null) {
570         contentHandler.endElement(uri, localName, qName);
571     }
572 }
573
574
575 /**
576  * Filter a character data event.
577  *
578  * @param ch An array of characters.
```

```
579     * @param start The starting position in the array.
580     * @param length The number of characters to use from the array.
581     * @exception org.xml.sax.SAXException The client may throw
582     *         an exception during processing.
583     */
584     public void characters (char ch[], int start, int length)
585         throws SAXException
586     {
587         if (contentHandler != null) {
588             contentHandler.characters(ch, start, length);
589         }
590     }
591
592
593     /**
594     * Filter an ignorable whitespace event.
595     *
596     * @param ch An array of characters.
597     * @param start The starting position in the array.
598     * @param length The number of characters to use from the array.
599     * @exception org.xml.sax.SAXException The client may throw
600     *         an exception during processing.
601     */
602     public void ignorableWhitespace (char ch[], int start, int length)
603         throws SAXException
604     {
605         if (contentHandler != null) {
606             contentHandler.ignorableWhitespace(ch, start, length);
607         }
608     }
609
610
611     /**
612     * Filter a processing instruction event.
613     *
614     * @param target The processing instruction target.
615     * @param data The text following the target.
616     * @exception org.xml.sax.SAXException The client may throw
617     *         an exception during processing.
618     */
619     public void processingInstruction (String target, String data)
620         throws SAXException
621     {
622         if (contentHandler != null) {
623             contentHandler.processingInstruction(target, data);
624         }
625     }
626
627
628     /**
629     * Filter a skipped entity event.
630     *
631     * @param name The name of the skipped entity.
```

```
632     * @exception org.xml.sax.SAXException The client may throw
633     *         an exception during processing.
634     */
635     public void skippedEntity (String name)
636         throws SAXException
637     {
638         if (contentHandler != null) {
639             contentHandler.skippedEntity(name);
640         }
641     }
642
643
644
645     //////////////////////////////////////
646     // Implementation of org.xml.sax.ErrorHandler.
647     //////////////////////////////////////
648
649
650     /**
651     * Filter a warning event.
652     *
653     * @param e The warning as an exception.
654     * @exception org.xml.sax.SAXException The client may throw
655     *         an exception during processing.
656     */
657     public void warning (SAXParseException e)
658         throws SAXException
659     {
660         if (errorHandler != null) {
661             errorHandler.warning(e);
662         }
663     }
664
665
666     /**
667     * Filter an error event.
668     *
669     * @param e The error as an exception.
670     * @exception org.xml.sax.SAXException The client may throw
671     *         an exception during processing.
672     */
673     public void error (SAXParseException e)
674         throws SAXException
675     {
676         if (errorHandler != null) {
677             errorHandler.error(e);
678         }
679     }
680
681
682     /**
683     * Filter a fatal error event.
684     *
```



```
685     * @param e The error as an exception.
686     * @exception org.xml.sax.SAXException The client may throw
687     *         an exception during processing.
688     */
689     public void fatalError (SAXParseException e)
690         throws SAXException
691     {
692         if (errorHandler != null) {
693             errorHandler.fatalError(e);
694         }
695     }
696
697
698
699     //////////////////////////////////////
700     // Internal methods.
701     //////////////////////////////////////
702
703
704     /**
705     * Set up before a parse.
706     *
707     * <p>Before every parse, check whether the parent is
708     * non-null, and re-register the filter for all of the
709     * events.</p>
710     */
711     private void setupParse ()
712     {
713         if (parent == null) {
714             throw new NullPointerException("No parent for filter");
715         }
716         parent.setEntityResolver(this);
717         parent.setDTDHandler(this);
718         parent.setContentHandler(this);
719         parent.setErrorHandler(this);
720     }
721
722
723
724     //////////////////////////////////////
725     // Internal state.
726     //////////////////////////////////////
727
728     private XMLReader parent = null;
729     private Locator locator = null;
730     private EntityResolver entityResolver = null;
731     private DTDHandler dtdHandler = null;
732     private ContentHandler contentHandler = null;
733     private ErrorHandler errorHandler = null;
734
735 }
736
737 // end of XMLFilterImpl.java
```

## 49.11 XMLReaderAdapter.java

Secuencia 513: org/xml/sax/helpers/XMLReaderAdapter.java

```
1  /*
2  * Copyright (c) 2000, 2005, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 */
25
26 // XMLReaderAdapter.java - adapt an SAX2 XMLReader to a SAX1 Parser
27 // http://www.saxproject.org
28 // Written by David Megginson
29 // NO WARRANTY! This class is in the public domain.
30 // $Id: XMLReaderAdapter.java,v 1.3 2004/11/03 22:53:09 jsuttor Exp $
31
32 package org.xml.sax.helpers;
33
34 import java.io.IOException;
35 import java.util.Locale;
36
37 import org.xml.sax.Parser;      // deprecated
38 import org.xml.sax.Locator;
39 import org.xml.sax.InputSource;
40 import org.xml.sax.AttributeList; // deprecated
41 import org.xml.sax.EntityResolver;
42 import org.xml.sax.DTDHandler;
43 import org.xml.sax.DocumentHandler; // deprecated
44 import org.xml.sax.ErrorHandler;
45 import org.xml.sax.SAXException;
46
47 import org.xml.sax.XMLReader;
48 import org.xml.sax.Attributes;
49 import org.xml.sax.ContentHandler;
50 import org.xml.sax.SAXNotSupportedException;
```

```

51
52
53 /**
54  * Adapt a SAX2 XMLReader as a SAX1 Parser.
55  *
56  * <blockquote>
57  * <em>This module, both source code and documentation, is in the
58  * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
59  * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
60  * for further information.
61  * </blockquote>
62  *
63  * <p>This class wraps a SAX2 {@link org.xml.sax.XMLReader XMLReader}
64  * and makes it act as a SAX1 {@link org.xml.sax.Parser Parser}. The XMLReader
65  * must support a true value for the
66  * http://xml.org/sax/features/namespace-prefixes property or parsing will fail
67  * with a {@link org.xml.sax.SAXException SAXException}; if the XMLReader
68  * supports a false value for the http://xml.org/sax/features/namespaces
69  * property, that will also be used to improve efficiency.</p>
70  *
71  * @since SAX 2.0
72  * @author David Megginson
73  * @see org.xml.sax.Parser
74  * @see org.xml.sax.XMLReader
75  */
76 public class XMLReaderAdapter implements Parser, ContentHandler
77 {
78
79
80     //////////////////////////////////////
81     // Constructor.
82     //////////////////////////////////////
83
84
85     /**
86      * Create a new adapter.
87      *
88      * <p>Use the "org.xml.sax.driver" property to locate the SAX2
89      * driver to embed.</p>
90      *
91      * @exception org.xml.sax.SAXException If the embedded driver
92      *         cannot be instantiated or if the
93      *         org.xml.sax.driver property is not specified.
94      */
95     public XMLReaderAdapter ()
96     throws SAXException
97     {
98         setup(XMLReaderFactory.createXMLReader());
99     }
100
101
102     /**
103      * Create a new adapter.

```

```

104      *
105      * <p>Create a new adapter, wrapped around a SAX2 XMLReader.
106      * The adapter will make the XMLReader act like a SAX1
107      * Parser.</p>
108      *
109      * @param xmlReader The SAX2 XMLReader to wrap.
110      * @exception java.lang.NullPointerException If the argument is null.
111      */
112     public XMLReaderAdapter (XMLReader xmlReader)
113     {
114         setup(xmlReader);
115     }
116
117
118
119     /**
120      * Internal setup.
121      *
122      * @param xmlReader The embedded XMLReader.
123      */
124     private void setup (XMLReader xmlReader)
125     {
126         if (xmlReader == null) {
127             throw new NullPointerException("XMLReader must not be null");
128         }
129         this.xmlReader = xmlReader;
130         qAtts = new AttributesAdapter();
131     }
132
133
134
135     //////////////////////////////////////
136     // Implementation of org.xml.sax.Parser.
137     //////////////////////////////////////
138
139
140     /**
141      * Set the locale for error reporting.
142      *
143      * <p>This is not supported in SAX2, and will always throw
144      * an exception.</p>
145      *
146      * @param locale the locale for error reporting.
147      * @see org.xml.sax.Parser#setLocale
148      * @exception org.xml.sax.SAXException Thrown unless overridden.
149      */
150     public void setLocale (Locale locale)
151         throws SAXException
152     {
153         throw new SAXNotSupportedException("setLocale not supported");
154     }
155
156

```

```
157  /**
158   * Register the entity resolver.
159   *
160   * @param resolver The new resolver.
161   * @see org.xml.sax.Parser#setEntityResolver
162   */
163  public void setEntityResolver (EntityResolver resolver)
164  {
165      xmlReader.setEntityResolver(resolver);
166  }
167
168
169  /**
170   * Register the DTD event handler.
171   *
172   * @param handler The new DTD event handler.
173   * @see org.xml.sax.Parser#setDTDHandler
174   */
175  public void setDTDHandler (DTDHandler handler)
176  {
177      xmlReader.setDTDHandler(handler);
178  }
179
180
181  /**
182   * Register the SAX1 document event handler.
183   *
184   * <p>Note that the SAX1 document handler has no Namespace
185   * support.</p>
186   *
187   * @param handler The new SAX1 document event handler.
188   * @see org.xml.sax.Parser#setDocumentHandler
189   */
190  public void setDocumentHandler (DocumentHandler handler)
191  {
192      documentHandler = handler;
193  }
194
195
196  /**
197   * Register the error event handler.
198   *
199   * @param handler The new error event handler.
200   * @see org.xml.sax.Parser#setErrorHandler
201   */
202  public void setErrorHandler (ErrorHandler handler)
203  {
204      xmlReader.setErrorHandler(handler);
205  }
206
207
208  /**
209   * Parse the document.
```

```

210      *
211      * <p>This method will throw an exception if the embedded
212      * XMLReader does not support the
213      * http://xml.org/sax/features/namespace-prefixes property.</p>
214      *
215      * @param systemId The absolute URL of the document.
216      * @exception java.io.IOException If there is a problem reading
217      *         the raw content of the document.
218      * @exception org.xml.sax.SAXException If there is a problem
219      *         processing the document.
220      * @see #parse(org.xml.sax.InputSource)
221      * @see org.xml.sax.Parser#parse(java.lang.String)
222      */
223     public void parse (String systemId)
224         throws IOException, SAXException
225     {
226         parse(new InputSource(systemId));
227     }
228
229     /**
230     * Parse the document.
231     *
232     * <p>This method will throw an exception if the embedded
233     * XMLReader does not support the
234     * http://xml.org/sax/features/namespace-prefixes property.</p>
235     *
236     * @param input An input source for the document.
237     * @exception java.io.IOException If there is a problem reading
238     *         the raw content of the document.
239     * @exception org.xml.sax.SAXException If there is a problem
240     *         processing the document.
241     * @see #parse(java.lang.String)
242     * @see org.xml.sax.Parser#parse(org.xml.sax.InputSource)
243     */
244     public void parse (InputSource input)
245         throws IOException, SAXException
246     {
247         setupXMLReader();
248         xmlReader.parse(input);
249     }
250
251     /**
252     * Set up the XML reader.
253     */
254     private void setupXMLReader ()
255         throws SAXException
256     {
257         xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
258         try {
259             xmlReader.setFeature("http://xml.org/sax/features/namespaces",
260                                 false);

```

```
263     } catch (SAXException e) {
264         // NO OP: it's just extra information, and we can ignore it
265     }
266     xmlReader.setContentHandler(this);
267 }
268
269
270
271 ///////////////////////////////////////////////////////////////////
272 // Implementation of org.xml.sax.ContentHandler.
273 ///////////////////////////////////////////////////////////////////
274
275
276 /**
277  * Set a document locator.
278  *
279  * @param locator The document locator.
280  * @see org.xml.sax.ContentHandler#setDocumentLocator
281  */
282 public void setDocumentLocator (Locator locator)
283 {
284     if (documentHandler != null)
285         documentHandler.setDocumentLocator(locator);
286 }
287
288
289 /**
290  * Start document event.
291  *
292  * @exception org.xml.sax.SAXException The client may raise a
293  *         processing exception.
294  * @see org.xml.sax.ContentHandler#startDocument
295  */
296 public void startDocument ()
297     throws SAXException
298 {
299     if (documentHandler != null)
300         documentHandler.startDocument();
301 }
302
303
304 /**
305  * End document event.
306  *
307  * @exception org.xml.sax.SAXException The client may raise a
308  *         processing exception.
309  * @see org.xml.sax.ContentHandler#endDocument
310  */
311 public void endDocument ()
312     throws SAXException
313 {
314     if (documentHandler != null)
315         documentHandler.endDocument();
316 }
```

```
316     }
317
318
319     /**
320      * Adapt a SAX2 start prefix mapping event.
321      *
322      * @param prefix The prefix being mapped.
323      * @param uri The Namespace URI being mapped to.
324      * @see org.xml.sax.ContentHandler#startPrefixMapping
325      */
326     public void startPrefixMapping (String prefix, String uri)
327     {
328     }
329
330
331     /**
332      * Adapt a SAX2 end prefix mapping event.
333      *
334      * @param prefix The prefix being mapped.
335      * @see org.xml.sax.ContentHandler#endPrefixMapping
336      */
337     public void endPrefixMapping (String prefix)
338     {
339     }
340
341
342     /**
343      * Adapt a SAX2 start element event.
344      *
345      * @param uri The Namespace URI.
346      * @param localName The Namespace local name.
347      * @param qName The qualified (prefixed) name.
348      * @param atts The SAX2 attributes.
349      * @exception org.xml.sax.SAXException The client may raise a
350      *      processing exception.
351      * @see org.xml.sax.ContentHandler#endDocument
352      */
353     public void startElement (String uri, String localName,
354                             String qName, Attributes atts)
355         throws SAXException
356     {
357         if (documentHandler != null) {
358             qAtts.setAttributes(atts);
359             documentHandler.startElement(qName, qAtts);
360         }
361     }
362
363
364     /**
365      * Adapt a SAX2 end element event.
366      *
367      * @param uri The Namespace URI.
368      * @param localName The Namespace local name.
```



```
369     * @param qName The qualified (prefixed) name.
370     * @exception org.xml.sax.SAXException The client may raise a
371     *         processing exception.
372     * @see org.xml.sax.ContentHandler#endElement
373     */
374     public void endElement (String uri, String localName,
375                             String qName)
376         throws SAXException
377     {
378         if (documentHandler != null)
379             documentHandler.endElement(qName);
380     }
381
382
383     /**
384     * Adapt a SAX2 characters event.
385     *
386     * @param ch An array of characters.
387     * @param start The starting position in the array.
388     * @param length The number of characters to use.
389     * @exception org.xml.sax.SAXException The client may raise a
390     *         processing exception.
391     * @see org.xml.sax.ContentHandler#characters
392     */
393     public void characters (char ch[], int start, int length)
394         throws SAXException
395     {
396         if (documentHandler != null)
397             documentHandler.characters(ch, start, length);
398     }
399
400
401     /**
402     * Adapt a SAX2 ignorable whitespace event.
403     *
404     * @param ch An array of characters.
405     * @param start The starting position in the array.
406     * @param length The number of characters to use.
407     * @exception org.xml.sax.SAXException The client may raise a
408     *         processing exception.
409     * @see org.xml.sax.ContentHandler#ignorableWhitespace
410     */
411     public void ignorableWhitespace (char ch[], int start, int length)
412         throws SAXException
413     {
414         if (documentHandler != null)
415             documentHandler.ignorableWhitespace(ch, start, length);
416     }
417
418
419     /**
420     * Adapt a SAX2 processing instruction event.
421     *
```

```

422     * @param target The processing instruction target.
423     * @param data The remainder of the processing instruction
424     * @exception org.xml.sax.SAXException The client may raise a
425     *         processing exception.
426     * @see org.xml.sax.ContentHandler#processingInstruction
427     */
428     public void processingInstruction (String target, String data)
429         throws SAXException
430     {
431         if (documentHandler != null)
432             documentHandler.processingInstruction(target, data);
433     }
434
435
436     /**
437     * Adapt a SAX2 skipped entity event.
438     *
439     * @param name The name of the skipped entity.
440     * @see org.xml.sax.ContentHandler#skippedEntity
441     * @exception org.xml.sax.SAXException Throwable by subclasses.
442     */
443     public void skippedEntity (String name)
444         throws SAXException
445     {
446     }
447
448
449
450     //////////////////////////////////////
451     // Internal state.
452     //////////////////////////////////////
453
454     XMLReader xmlReader;
455     DocumentHandler documentHandler;
456     AttributesAdapter qAtts;
457
458
459
460     //////////////////////////////////////
461     // Internal class.
462     //////////////////////////////////////
463
464
465     /**
466     * Internal class to wrap a SAX2 Attributes object for SAX1.
467     */
468     final class AttributesAdapter implements AttributeList
469     {
470         AttributesAdapter ()
471         {
472         }
473
474

```

```
475     /**
476      * Set the embedded Attributes object.
477      *
478      * @param The embedded SAX2 Attributes.
479      */
480     void setAttributes (Attributes attributes)
481     {
482         this.attributes = attributes;
483     }
484
485
486     /**
487      * Return the number of attributes.
488      *
489      * @return The length of the attribute list.
490      * @see org.xml.sax.AttributeList#getLength
491      */
492     public int getLength ()
493     {
494         return attributes.getLength();
495     }
496
497
498     /**
499      * Return the qualified (prefixed) name of an attribute by position.
500      *
501      * @return The qualified name.
502      * @see org.xml.sax.AttributeList#getName
503      */
504     public String getName (int i)
505     {
506         return attributes.getQName(i);
507     }
508
509
510     /**
511      * Return the type of an attribute by position.
512      *
513      * @return The type.
514      * @see org.xml.sax.AttributeList#getType(int)
515      */
516     public String getType (int i)
517     {
518         return attributes.getType(i);
519     }
520
521
522     /**
523      * Return the value of an attribute by position.
524      *
525      * @return The value.
526      * @see org.xml.sax.AttributeList#getValue(int)
527      */
```

```

528     public String getValue (int i)
529     {
530         return attributes.getValue(i);
531     }
532
533
534     /**
535      * Return the type of an attribute by qualified (prefixed) name.
536      *
537      * @return The type.
538      * @see org.xml.sax.AttributeList#getType(java.lang.String)
539      */
540     public String getType (String qName)
541     {
542         return attributes.getType(qName);
543     }
544
545
546     /**
547      * Return the value of an attribute by qualified (prefixed) name.
548      *
549      * @return The value.
550      * @see org.xml.sax.AttributeList#getValue(java.lang.String)
551      */
552     public String getValue (String qName)
553     {
554         return attributes.getValue(qName);
555     }
556
557     private Attributes attributes;
558 }
559
560 }
561
562 // end of XMLReaderAdapter.java

```

## 49.12 XMLReaderFactory.java

Secuencia 514: org/xml/sax/helpers/XMLReaderFactory.java

```

1  /*
2  * Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
3  * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
4  *
5  *
6  *
7  *
8  *
9  *
10 *
11 *
12 *
13 *
14 *

```

```
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  */
25
26  // XMLReaderFactory.java - factory for creating a new reader.
27  // http://www.saxproject.org
28  // Written by David Megginson
29  // and by David Brownell
30  // NO WARRANTY! This class is in the Public Domain.
31  // $Id: XMLReaderFactory.java,v 1.2.2.1 2005/07/31 22:48:08 jeffsuttor Exp $
32
33  package org.xml.sax.helpers;
34  import java.io.BufferedReader;
35  import java.io.InputStream;
36  import java.io.InputStreamReader;
37  import org.xml.sax.XMLReader;
38  import org.xml.sax.SAXException;
39
40
41  /**
42   * Factory for creating an XML reader.
43   *
44   * <blockquote>
45   * <em>This module, both source code and documentation, is in the
46   * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
47   * See <a href='http://www.saxproject.org'>http://www.saxproject.org</a>
48   * for further information.
49   * </blockquote>
50   *
51   * <p>This class contains static methods for creating an XML reader
52   * from an explicit class name, or based on runtime defaults:</p>
53   *
54   * <pre>
55   * try {
56   *     XMLReader myReader = XMLReaderFactory.createXMLReader();
57   * } catch (SAXException e) {
58   *     System.err.println(e.getMessage());
59   * }
60   * </pre>
61   *
62   * <p><strong>Note to Distributions bundled with parsers:</strong>
63   * You should modify the implementation of the no-arguments
64   * <em>createXMLReader</em> to handle cases where the external
65   * configuration mechanisms aren't set up. That method should do its
66   * best to return a parser when one is in the class path, even when
67   * nothing bound its class name to <code>org.xml.sax.driver</code> so
```

```

68  * those configuration mechanisms would see it.</p>
69  *
70  * @since SAX 2.0
71  * @author David Megginson, David Brownell
72  * @version 2.0.1 (sax2r2)
73  */
74  final public class XMLReaderFactory
75  {
76      /**
77       * Private constructor.
78       *
79       * <p>This constructor prevents the class from being instantiated.</p>
80       */
81      private XMLReaderFactory ()
82      {
83      }
84
85      private static final String property = "org.xml.sax.driver";
86      private static SecuritySupport ss = new SecuritySupport();
87
88      private static String _clsFromJar = null;
89      private static boolean _jarread = false;
90      /**
91       * Attempt to create an XMLReader from system defaults.
92       * In environments which can support it, the name of the XMLReader
93       * class is determined by trying each these options in order, and
94       * using the first one which succeeds:</p> <ul>
95       *
96       * <li>If the system property <code>org.xml.sax.driver</code>
97       * has a value, that is used as an XMLReader class name. </li>
98       *
99       * <li>The JAR "Services API" is used to look for a class name
100      * in the <em>META-INF/services/org.xml.sax.driver</em> file in
101      * jarfiles available to the runtime.</li>
102      *
103      * <li>SAX parser distributions are strongly encouraged to provide
104      * a default XMLReader class name that will take effect only when
105      * previous options (on this list) are not successful.</li>
106      *
107      * <li>Finally, if {@link ParserFactory#makeParser()} can
108      * return a system default SAX1 parser, that parser is wrapped in
109      * a {@link ParserAdapter}. (This is a migration aid for SAX1
110      * environments, where the <code>org.xml.sax.parser</code> system
111      * property will often be usable.) </li>
112      *
113      * </ul>
114      *
115      * <p>In environments such as small embedded systems, which can not
116      * support that flexibility, other mechanisms to determine the default
117      * may be used. </p>
118      *
119      * <p>Note that many Java environments allow system properties to be
120      * initialized on a command line. This means that <em>in most cases</em>

```

```

121     * setting a good value for that property ensures that calls to this
122     * method will succeed, except when security policies intervene.
123     * This will also maximize application portability to older SAX
124     * environments, with less robust implementations of this method.
125     * </p>
126     *
127     * @return A new XMLReader.
128     * @exception org.xml.sax.SAXException If no default XMLReader class
129     *         can be identified and instantiated.
130     * @see #createXMLReader(java.lang.String)
131     */
132     public static XMLReader createXMLReader ()
133         throws SAXException
134     {
135         String      className = null;
136         ClassLoader cl = ss.getContextClassLoader();
137
138         // 1. try the JVM-instance-wide system property
139         try {
140             className = ss.getProperty(property);
141         }
142         catch (RuntimeException e) { /* continue searching */ }
143
144         // 2. if that fails, try META-INF/services/
145         if (className == null) {
146             if (!_jarread) {
147                 _jarread = true;
148                 String      service = "META-INF/services/" + property;
149                 InputStream in;
150                 BufferedReader reader;
151
152                 try {
153                     if (cl != null) {
154                         in = ss.getResourceAsStream(cl, service);
155
156                         // If no provider found then try the current ClassLoader
157                         if (in == null) {
158                             cl = null;
159                             in = ss.getResourceAsStream(cl, service);
160                         }
161                     } else {
162                         // No Context ClassLoader, try the current ClassLoader
163                         in = ss.getResourceAsStream(cl, service);
164                     }
165
166                     if (in != null) {
167                         reader = new BufferedReader (new InputStreamReader (in, "UTF8"));
168                         _clsFromJar = reader.readLine ();
169                         in.close ();
170                     }
171                 } catch (Exception e) {
172                 }
173             }

```

```

174         className = _clsFromJar;
175     }
176
177     // 3. Distro-specific fallback
178     if (className == null) {
179 // BEGIN DISTRIBUTION-SPECIFIC
180
181         // EXAMPLE:
182         // className = "com.example.sax.XmlReader";
183         // or a $JAVA_HOME/jre/lib/*properties setting...
184         className = "com.sun.org.apache.xerces.internal.parsers.SAXParser";
185
186 // END DISTRIBUTION-SPECIFIC
187     }
188
189     // do we know the XMLReader implementation class yet?
190     if (className != null)
191         return loadClass (cl, className);
192
193     // 4. panic -- adapt any SAX1 parser
194     try {
195         return new ParserAdapter (ParserFactory.makeParser ());
196     } catch (Exception e) {
197         throw new SAXException ("Can't create default XMLReader; "
198             + "is system property org.xml.sax.driver set?");
199     }
200 }
201
202
203 /**
204  * Attempt to create an XML reader from a class name.
205  *
206  * <p>Given a class name, this method attempts to load
207  * and instantiate the class as an XML reader.</p>
208  *
209  * <p>Note that this method will not be usable in environments where
210  * the caller (perhaps an applet) is not permitted to load classes
211  * dynamically.</p>
212  *
213  * @return A new XML reader.
214  * @exception org.xml.sax.SAXException If the class cannot be
215  *         loaded, instantiated, and cast to XMLReader.
216  * @see #createXMLReader()
217  */
218 public static XMLReader createXMLReader (String className)
219     throws SAXException
220 {
221     return loadClass (ss.getContextClassLoader(), className);
222 }
223
224 private static XMLReader loadClass (ClassLoader loader, String className)
225     throws SAXException
226 {

```



```
227     try {
228         return (XMLReader) NewInstance.newInstance (loader, className);
229     } catch (ClassNotFoundException e1) {
230         throw new SAXException("SAX2 driver class " + className +
231                                " not found", e1);
232     } catch (IllegalAccessException e2) {
233         throw new SAXException("SAX2 driver class " + className +
234                                " found but cannot be loaded", e2);
235     } catch (InstantiationException e3) {
236         throw new SAXException("SAX2 driver class " + className +
237                                " loaded but cannot be instantiated (no empty public constructor?)",
238                                e3);
239     } catch (ClassCastException e4) {
240         throw new SAXException("SAX2 driver class " + className +
241                                " does not implement XMLReader", e4);
242     }
243 }
244 }
```